— INTERNSHIP REPORT —

CLASSIFYING COVERING SPACES IN HOMOTOPY TYPE THEORY[a]

Hugo Salou
ENS de Lyon
Département Informatique

Internship supervisors:
Samuel Mimram et Émile Oleon
LIX, École Polytechnique
Équipe Cosynus

01/06/2025 – 25/07/2025

[a]Original title: « Preuve formelle des variétés polyédrales de Poincaré en théorie des types homotopiques »

Contents

1. Introduction.

Algebraic topology is the study of topological spaces through algebra. We may compare two spaces by studying *invariants*. A simple invariant is the number of connected components in the space.

Classifying spaces "*up to equality*" is very restrictive. When we consider the usual example of the coffee mug that can be deformed into a torus, we want to consider that the two spaces are identical. This tells us that we should consider spaces "*up to continuous deformation*," we will give a more precise definition in section 2.

The goal of this internship was to formally prove (*i.e.* in a proof assistant) a classical result in algebraic topology: there is a one to one correspondence between covering spaces and subgroups of the fundamental group. All of the definitions will be give in section 2. However, the main idea is: by studying subgroups, we get geometric insights.

To prove this result, we use *Homotopy Type Theory* (*HoTT*). This framework links *geometry* and *logic* in one coherent paradigm based on *Type Theory* (used in most proof assistants). It allows us to very simply reason about many of the key concepts of algebraic topology. Instead of thinking of types as logical propositions (as it usual in Type Theory), we think of them as topological spaces. We will discuss more about HoTT in section 3.

Agda is the proof assistant used during this internship. When using proof assistants like Rocq or Lean, we expect to use tactics in order to prove a result (so, a type, as it is usual in Type Theory). These tactics generate a term of the corresponding type. However, Agda doesn't go in this direction. Instead, we write terms directly with a Haskell-like syntax. The *Cubical* version of Agda allows us to use *Cubical Homotopy Type Theory*, a variant of the one presented in the *HoTT Book* [Uni13] (the main reference for Homotopy Type Theory). A more complete presentation of Agda will be given in subsection 3.5.

In the appendices, you will find some technical background in Category Theory (appendix B) and some more context about this internship (appendix A).

Acknowledgments. For introducing me to the world of Algebraic Topology and Homotopy Type Theory, I am very grateful to Samuel MIMRAM and Émile OLEON for being my internship supervisors. I would also like to thank the other members of the *Cosynus* team for kindly welcoming me.

Related Work. In this internship, we give an Agda-verified version of the proof in [MO25]. Covering spaces in HoTT were first introduced by [HH18]; then, [WMP24] gave a algebraic-topology-focused proof of the classification of covering spaces. The proof of the Galois Correspondence given here (as the one in [MO25]) is shorter, more conceptual and more easily generalizable to n-coverings (*c.f.* [MO25]).

2. Some elements of Algebraic Topology.

In this section, we will give an introduction to algebraic topology. The main source for this section is [Hat02, Chapters 1 & 2].

In this section, we will write $\mathbf{I}$ for $[0,1]$–the unit interval. Let X be a topological space, that is, some space with a notion of *openness* and thus *continuity*. In this section, we only deal with continuous functions, so a "map" will refer to a continuous function.

2.1. Paths and Loops.

Let us start with a definition of paths and loops.

Definition 1.
- ▹ A ***path*** from x to y (in the topological space X) is a map $p : \mathbf{I} \to X$ such that $p(0) = x$ and $p(1) = y$.
- ▹ A ***loop*** around x is a path from x to x.

We will write $p : x \rightsquigarrow y$ when p is a path from x to y (in some implied space X). We expect to be able to *concatenate* paths, and also to *reverse* them.

Definition 2. Let $p : x \rightsquigarrow y$ and $q : y \rightsquigarrow z$ be paths.

- ▹ The ***reversed*** or ***inverse*** of path p is the path p^{-1} defined by:

$$p^{-1}(t) := p(1-t).$$

- ▹ The ***concatenation*** of paths p and q is the path $p \cdot q$ defined by :

$$p \cdot q(t) := \begin{cases} p(2t) & \text{if } 0 \le t \le \frac{1}{2} \\ q(2t-1) & \text{if } \frac{1}{2} \le t \le 1. \end{cases}$$

- ▹ The ***constant loop*** at point x is the loop refl_x defined by:

$$\mathsf{refl}_x(t) := x.$$

We can take the reverse of any paths but we need that the common endpoint matches when we want to concatenate paths. The notation refl_x may seem unusual when talking about paths, but it'll make more sense after section 3.

We'd love to have some nice properties with the concatenation and inverse: for example, associativity of concatenation, or that refl_x behaves like a neutral element for concatenation. However, *in general*, we do not have

$$(p \cdot q) \cdot r \neq p \cdot (q \cdot r).$$

This is a *timing issue*. For the first path, we spend the first quarter travelling along p, then the next quarter along q, and finally the rest along

r. For the second path, we spend the first half travelling along p and then a quarter along q and finally the rest along r.

Another example is $p \cdot p^{-1}$: this path "behaves" like the loop refl_x, but it's not equal to refl_x. And for this example, it's more than a timing issue, we need to *smoothly deform* the path p, bringing closer the endpoint y to x until we get refl_x.

Strict equality is not the right way to compare paths. The right way is *equality up to smooth deformation*, also known as a *homotopy*.

Definition 3. A ***homotopy*** from a path $p : x \rightsquigarrow y$ to a path $q : x \rightsquigarrow y$ is a map

$$h : \mathbf{I} \times \mathbf{I} \to X,$$

such that: for all $t \in \mathbf{I}$,

1. $h(0, t) = p(t)$,
2. $h(1, t) = q(t)$,
3. $h(t, 0) = x$,
4. $h(t, 1) = y$.

We say that p and q are ***homotopic***, written $p \approx_p q$ when there exists a homotopy from p to q.

In the definition above, the conditions are *boundary conditions*: that is, the initial slice (when the first parameter is 0) is p, the final slice is q and all the slices are paths from x to y. The relation "homotopic to" is an equivalence relation.

Uncurrying the definition of a homotopy, we have, in some sense, a path $h : p \rightsquigarrow q$ in the space of paths between x and y. To be very formal, we'd need to define a topology on the set of paths between x and y, which we don't want to do. However, it is an important idea to keep in mind.

Now that we have defined a new way of comparing paths, we need to make sure that

1. the notions of concatenation and inverse are defined up to homotopy,
2. and we get the nice properties we expect from concatenation and inverses.

Luckily, we do.

Lemma 1. Let $p, q : u \rightsquigarrow v$ and $r, s : v \rightsquigarrow w$ and $t : w \rightsquigarrow x$. We have:

- if $p \approx_p q$ then $p^{-1} \approx_p q^{-1}$;
- if $p \approx_p q$ and $r \approx_p s$ then $p \cdot r \approx_p q \cdot s$;
- $(p \cdot r) \cdot t \approx_p p \cdot (r \cdot t)$;
- $p \cdot \mathsf{refl}_y \approx_p p \approx_p \mathsf{refl}_x \cdot p$;
- $p \cdot p^{-1} \approx_p \mathsf{refl}_x$;
- $p^{-1} \cdot p \approx_p \mathsf{refl}_y$. □

With these property in mind, we can now introduce the fundamental group.

2.2. Fundamental Group.

Definition 4. Given a topological space X with $x \in X$ a point (we say that (X, x) is a ***pointed topological space***), its ***fundamental group*** $\pi_1(X, x)$ is the set of homotopy classes of loops at x in space X.

The fundamental group really has the structure of a group, thanks to lemma 1.

The fundamental group is one of those *invariants* discussed in the introduction. Let us give a simple example of how the fundamental group can be used to compare topological spaces.

Example 1. Consider the space $\mathbb{R}^2$ with the usual topology. The group $\pi_1(\mathbb{R}^2, (1, 0))$ is trivial: any loop is homotopic to $\mathsf{refl}_{(1,0)}$. You can think of scaling any loop at $(1, 0)$ until it vanishes into a constant loop. As we didn't tear the loop, our deformation was smooth, *i.e.*, a homotopy.

Example 2. Consider the subspace

$$V := \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 \geq 1\}$$

of $\mathbb{R}^2$. It corresponds to $\mathbb{R}^2$ without a unit open disk. We have that $\pi_1(V, (1, 0))$ is isomorphic to $\mathbb{Z}$. Let us see why in 3 steps.

1. There is a loop at point $(1, 0)$ that goes once "around the hole in V": it is the loop p defined by $p(t) := (\cos 2\pi t, \sin 2\pi t)$.
2. Any two different powers[b] of loop p are non homotopic. For example, p is not homotopic to the constant $\mathsf{refl}_{(1,0)}$ as it'd require cutting p to "go through" the hole in V.
3. Given any loop at $(1, 0)$, we can project it into a loop such that every point is at distance 1 from the origin. Then, we can smoothly deform the loop into p^n (where n is the *signed* number of times you go around the hole).

This tells us that $\pi_1(V, (1, 0))$ is freely generated by the homotopy class of p and thus is isomorphic to $\mathbb{Z}$.

The two spaces $\mathbb{R}^2$ and V are topologically different: they have a different fundamental group. It makes sense because $\mathbb{R}^2$ is topologically equivalent to a single point, and V is topologically equivalent to the unit circle $\mathbb{S}^1$. Only studying the number of connected components wouldn't have given us this geometric insight.

Change of base point. The ***base point*** of a pointed space (X, x) is x. What happens to the fundamental group when we change the base point? The answer is: not a lot, as long as the two base points have a path between them.

Proposition 1. Let $p : x \rightsquigarrow y$ be a path. Then,

$$\begin{aligned} \pi_1(X, x) &\longrightarrow \pi_1(X, y) \\ [\, q \,] &\longmapsto [\, p^{-1} \cdot q \cdot p \,] \end{aligned}$$

is a group isomorphism, where $[\, q \,]$ is the homotopy class of a path q. □

So, when X is non-empty and ***path-connected*** (that is, for any two pairs of points, there is a path between them), there is no need to specify a base point.

When there is no path between x and y, the fundamental groups could be *very different*. For example, if we consider embedding $\mathbb{R}^2$ and V on two parallel planes in $\mathbb{R}^3$ and considering the union of those two planes, then the fundamental group in the $\mathbb{R}^2$-plane is still trivial and other one is still isomorphic to $\mathbb{Z}$.

Functoriality. The π_1 "operation" acts on pointed spaces, but also acts on *pointed maps*: that is, a basepoint-preserving map between pointed spaces.[c] We say that π_1 is ***functorial***.

Proposition 2. Let (X, x) and (Y, y) be pointed spaces. A pointed map

$$f : (X, x) \to (Y, y)$$

induces a group homomorphism

$$\begin{aligned} \pi_1(f) : \pi_1(X, x) &\longrightarrow \pi_1(Y, y) \\ [\, p \,] &\longmapsto [\, f \circ p \,]. \end{aligned}$$

Thus, a pointed ***homeomorphism*** (a bijective continuous function whose inverse is also continuous) induces a group isomorphism.

Homeomorphic spaces have the same fundamental group. It would

[b] That is, repeated concatenation of p, or p^{-1} if the power is negative, as usual when manipulating groups.

[c] More explicitly, $f : (X, x) \to (Y, y)$ is a ***pointed map*** if $f : X \to Y$ is continuous and if $f(x) = y$.

seem reasonable to compare topological spaces *up to homeomorphism* but, as with path equality, it is too restrictive. For example, $\mathbb{R}^2$ and $\{0\}$ are topologically equivalent, but aren't homeomorphic (simply for cardinality reasons). We need to use *homotopy equivalences*.

2.3. Homotopy equivalence.

Let us first define a *homotopy of maps*.

Definition 5. A ***homotopy (of maps)*** from a map $f : X \to Y$ to a map $g : X \to Y$ is a map

$$h : X \times \mathbf{I} \to Y,$$

such that: for all $x \in X$,

1. $h(x, 0) = f(x)$,
2. $h(x, 1) = g(x)$.

When f and g are ***map-homotopic*** (that is, there exists a homotopy of maps between them), we write $f \approx g$. It is an equivalence relation.

We already introduced some notation for homotopies of paths: $p \approx_{\mathrm{p}} q$. Is it the same as $p \approx q$? No, in path homotopy, we require that the endpoints of all slices are x and y. For example, any path p is map-homotopic to the constant loop refl_x.

Now that we have a new way of comparing functions, we need to make sure that composition and inverses (when they exist) are compatible with map-homotopy.

Lemma 2. Given $f, u : X \to Y$ and $g, v : Y \to Z$, such that $f \approx u$ and $g \approx v$, we have that:

- $f \circ g \approx u \circ v$;
- $f^{-1} \approx u^{-1}$ if f and u are homeomorphisms.

Then we can finally define the notion of *homotopy equivalences*.

Definition 6. A ***homotopy equivalence*** between X and Y is a two maps $f : X \to Y$ and $g : Y \to X$ such that:

$$f \circ g \approx \mathsf{id}_Y \qquad \text{and} \qquad g \circ f \approx \mathsf{id}_X.$$

We write $X \simeq Y$ when X and Y are ***homotopy equivalent***. It is an equivalence relation.

Example 3. The spaces $\mathbb{R}^2$ and $\{0\}$ are homotopy equivalent. We define:

- $f : \mathbb{R}^2 \to \{0\}, x \mapsto 0$;
- $g : \{0\} \to \mathbb{R}^2, 0 \mapsto (0, 0)$.

On the one hand, we easily have that $f \circ g = \mathsf{id}_{\{0\}}$. On the other hand, we need to provide a homotopy between $x \in \mathbb{R}^2 \mapsto (0, 0) \in \mathbb{R}^2$ and $\mathsf{id}_{\mathbb{R}^2}$, so a map

$$h : \mathbb{R}^2 \times \mathbf{I} \to \mathbb{R}^2,$$

such that the initial slice is $x \mapsto (0, 0)$ and the final one is $x \mapsto x$. We can define it, for example, with

$$h((x, y), t) := (tx, ty).$$

This can be generalized to $\mathbb{R}^n \simeq \{0\}$ for any $n \in \mathbb{N}$.

Example 4. Recalling the definition of V from example 2:

$$V := \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 \geq 1\}.$$

Then V is homotopy equivalent to $\mathbb{S}^1$. Here is a homotopy equivalence:

- the map $f : \mathbb{S}^1 \to V$ is the inclusion map;
- the map $g : V \to \mathbb{S}^1$ is $\vec{v} \mapsto \vec{v}/||\vec{v}||$;
- we have $g \circ f = \mathsf{id}_{\mathbb{S}^1}$;
- we provide $h : (\vec{v}, t) \mapsto (1 - t)\vec{v} + t\vec{v}/||\vec{v}||$ a homotopy between $f \circ g : \vec{v} \to \vec{v}/||\vec{v}||$ and id_V.

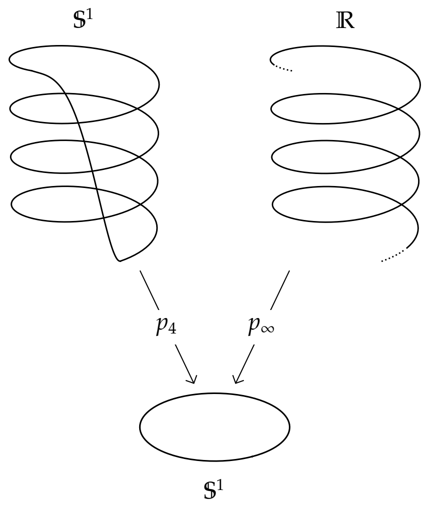


Figure 1 | *4-sheeted cover and universal cover of* $\mathbb{S}^1$

To justify the use of homotopy equivalences, we should give some results about homotopy equivalences and the fundamental group.

Proposition 3. If $f : (X, x) \to (Y, y)$ is a pointed homotopy equivalence, then the induced group homomorphism $\pi_1(f) : \pi_1(X, x) \to \pi_1(Y, y)$ is an isomorphism. □

This concludes the subsection on homotopy equivalences.

2.4. Covering spaces.

We will start by defining what a *covering space* is and then give some examples.

Definition 7. Let (X, x) be a pointed topological space. A ***(pointed) covering space*** is a space $\tilde{X}$ with a pointed map $p : (\tilde{X}, \tilde{x}) \to (X, x)$ with the following property: every point $\tilde{y} \in \tilde{X}$ has an open neighborhood U which is ***evenly covered***, that means the preimage $p^{-1}(U)$ is the union of disjoint open sets in $\tilde{X}$, each of which is mapped homeomorphically onto U by p.

The disjoints open sets composing the preimage $p^{-1}(U)$ are called ***sheets*** of $\tilde{X}$ over U.

Covering spaces give us a "local approximation" of our space. When dealing with complex spaces, studying this local lookalike give us some geometric insight, and hiding the complexity of the global behavior. They're also useful in other fields like Algebraic Geometry and Differential Geometry for these reasons.

Here are some examples of covering spaces. We sometimes leave implicit the chosen basepoint.

Example 5. Given a topological space X and some discrete space D, the space $\tilde{X} := X \times D$ with the map $p : (x, d) \mapsto x$ is called a ***trivial covering space*** of X.

Example 6. For any $n \in \mathbb{N}$, we can construct a covering space of $\mathbb{S}^1$: we choose $\tilde{X}$ to be $\mathbb{S}^1$, which we think as the subset of $\mathbb{C}$ whose points have a magnitude of 1, and we choose the map

$$\begin{aligned} p_n : \mathbb{S}^1 &\longrightarrow \mathbb{S}^1 \\ z &\longmapsto z^n. \end{aligned}$$

This corresponds to a covering of $\mathbb{S}^1$ with n sheets. Figure 1 shows this covering space for $n = 4$ (left one).

Figure 1 shows a smooth deformation of $\tilde{X}$ such that applying p corresponds to "flattening" that space on the unit circle $\mathbb{S}^1$.

Example 7. Another covering space of $\mathbb{S}^1$ is $\mathbb{R}$ with the map $p_\infty : x \mapsto$

$e^{2i\pi x}$. We call it the *universal cover* of $\mathbb{S}^1$ for reasons we will explain later. Figure 1 shows this covering space (right one). This universal cover is very important: it shows us what the space (here $\mathbb{S}^1$) but with only trivial loops, up to homotopy. In our case, it looks like unwinding the circle into an infinitely long line–$\mathbb{R}$.

One important feature of covering spaces is the ***lifting property***. Covering spaces are usually thought as being "geometrically above" the topological space considered, as it is in figure 1. Given a map whose codomain is X, we can "lift" it as a map whose codomain is $\tilde{X}$ such that projecting it back onto X give us the expected result.

Definition 8. A ***lift*** of $f : Y \to X$ along $p : \tilde{X} \to X$ is a map $\tilde{f} : Y \to \tilde{X}$ such that the following diagram commutes:

$$\begin{tikzcd} & \tilde{X} \arrow[d, "p"] \\ Y \arrow[ur, dashed, "\tilde{f}"] \arrow[r, "f"'] & X, \end{tikzcd}$$

i.e. $p \circ \tilde{f} = f$.

Lifts have a really powerful property:

Lemma 3 (Unique lifting property)**.** Given two lifts, if they agree on one point, then they agree on the whole connected component. □

Thus, considering path-connected pointed spaces, we only need to follow the basepoint to know the lift's whole behavior.

Paths and homotopy can thus be uniquely lifted, taking $Y = \mathbf{I}$ and $Y = \mathbf{I} \times \mathbf{I}$, the unit interval and unit square respectively. However, for these two cases, we also have existence.

Lemma 4 (Path/Homotopy lifting property)**.** Let $\tilde{X}$ be a cover of X.

1. Given a path $q : x \rightsquigarrow y$ in X, there exists a unique path $\tilde{q}$ such that $\tilde{q}(0) = x$ and $p \circ \tilde{q} = q$.
2. Furthermore, if $q \simeq_{\mathrm{p}} r$ then $\tilde{q} \simeq_{\mathrm{p}} \tilde{r}$, and in particular $\tilde{q}(1) = \tilde{r}(1)$.

We can define *morphisms between pointed covering spaces*. Thus, for a given topological space X, the collection of pointed covering spaces of X with morphisms between them forms a ***category*** named $\mathbf{Cov}(X, x)$ (*i.e.* composition of morphisms of cover and identities behave as we expect[d]; see appendix B for more details).

Definition 9. Given $(\tilde{X}, \tilde{x}, \tilde{p})$ and $(\bar{X}, \bar{x}, \bar{p})$ two covering spaces of X, a ***morphism of covering spaces*** from $(\tilde{X}, \tilde{x}, \tilde{p})$ to $(\bar{X}, \bar{x}, \bar{p})$ is a map $f : (\tilde{X}, \tilde{x}) \to (\bar{X}, \bar{x})$ such that the following diagram commutes:

$$\begin{tikzcd} \tilde{X}, \tilde{x} \arrow[rr, "f"] \arrow[dr, "\tilde{p}"'] & & \bar{X}, \bar{x} \arrow[dl, "\bar{p}"] \\ & X, x & \end{tikzcd}$$

i.e. such that $\bar{p} \circ f = \tilde{p}$.

We can now define what *universal covering* means (and fully understand example 7).

Definition 10. A covering $(\tilde{X}, \tilde{p})$ of X is said to be ***universal*** if for any covering $(\bar{X}, \bar{p})$ of X there is a morphism from $(\tilde{X}, \tilde{p})$ to $(\bar{X}, \bar{p})$.

We can safely say "*the*" universal cover as it is unique up to isomorphism. This results from the fact that morphisms of covering spaces are lifts.

Lemma 5. Given two universal covers, they are isomorphic.

Proof. Let $\tilde{X}$ and $\bar{X}$ be two universal covers. By definition, there are morphisms $f : \tilde{X} \to \bar{X}$ and $g : \bar{X} \to \tilde{X}$ such that

$$\begin{tikzcd} \tilde{X}, \tilde{x} \arrow[r, "f"] \arrow[dr, "\tilde{p}"'] & \bar{X}, \bar{x} \arrow[r, "g"] \arrow[d, "\bar{p}"] & \tilde{X}, \tilde{x} \arrow[dl, "\tilde{p}"] \\ & X, x & \end{tikzcd}$$

commutes. Then, by rearranging, we obtain the following commutative diagram

$$\begin{tikzcd} & \tilde{X}, \tilde{x} \arrow[d, "\tilde{p}"] \\ \tilde{X}, \tilde{x} \arrow[ur, "g \circ f"] \arrow[r, "\tilde{p}"'] & X, x \end{tikzcd}$$

which, by lemma 3 on $\tilde{X}$, implies $g \circ f = \mathrm{id}_{\tilde{X}}$, as $g \circ f(\tilde{x}) = \tilde{x}$. Similarly, by applying the lemma on $\bar{X}$, we have $f \circ g = \mathrm{id}_{\bar{X}}$. □

We also prove an important result: as soon as we have two morphisms $f : \tilde{X} \to \bar{X}$ and $g : \bar{X} \to \tilde{X}$ then $\tilde{X}$ and $\bar{X}$ are isomorphic (for any covering spaces). This give us an important lemma:

Lemma 6. Let $\mathrm{Cov}(X)$ be the set of covering spaces of X. We order $\mathrm{Cov}(X)$ by the relation $\preceq$: we have $\tilde{X} \preceq \bar{X}$ if and only if there is a morphism from $\tilde{X}$ to $\bar{X}$. This forms a preoder (reflexive and transitive) and thus we obtain an ordering on the isomorphism classes of covering spaces of X.

Proof. Reflexivity with $\mathrm{id}_{\tilde{X}}$, transitivity with composition and symmetry with the above remark. □

Under stronger hypotheses, we have an even better structure: a ***complete lattice***. The supremum of this lattice is the universal cover. It always exists as given by the following construction.

Definition 11. We say that a space X is:

- ***locally path-connected*** when, for every point x and every neighborhood U, there exists a smaller neighborhood $V \subseteq U$ of x that is path-connected;
- ***simply-connected*** when it is path-connected and has a trivial fundamental group, *i.e.* every loop is homotopic to refl;
- ***semi-locally simply connected*** when, for every point x and every neighborhood U, any loop in U is homotopic to refl_x in X.

Proposition 4. Let (X, x) be a pointed space. Suppose X is:

- path-connected ;
- locally path-connected ;
- semi-locally simply connected.

Then the set

$$\big\{ [\,q\,] \,\big|\, q : x \rightsquigarrow y \text{ for some point } y \big\}$$

is the universal cover of (X, x) with the map $[\,q\,] \mapsto q(1)$. □

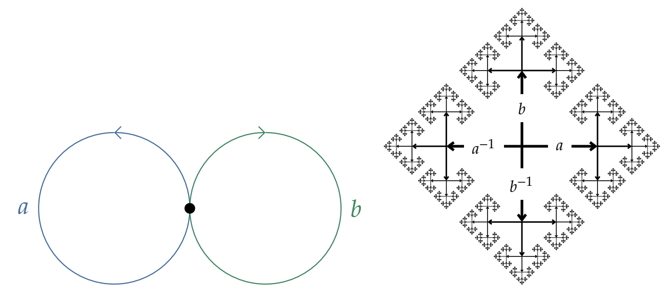
Using this construction, we can construct a fractal-like universal covering for the wedge of two circles.

Example 8. The ***wedge of two circles***, written $\mathbb{S}^1 \vee \mathbb{S}^1$, is the shape shown in figure 2(a). Its fundamental group is the free group with two generators a and b (those correspond to the two loops shown in figure 2(a)). Its universal cover is the fractal tree shown in figure 2(b). To see why, we can fold the loops a and b to obtain $\mathbb{S}^1 \vee \mathbb{S}^1$ and the preimage of any neighborhood (for a small enough one) is discrete, thus a covering. It is universal thanks to the following result.

The "usual" definition of the universal cover is a cover with trivial fundamental group–it is an equivalent definition.

Proposition 5. A covering is universal iff it has a trivial fundamental group. □

[d] That is: associativity of composition and identity as neutral elements of composition.



(a) Wedge of two circles (b) Universal cover

Figure 2 | *Wedge of two circles, $\mathbb{S}^1 \vee \mathbb{S}^1$, and its universal cover*

This result will be a simple consequence of the Galois Correspondence.

2.5. The Galois Correspondence.

We have now defined all the notions we need to understand the ***Galois Correspondence***: fundamental groups and covering spaces are *very* closely related. This is the result that was proven during this internship.

This result is analogue to the Galois Correspondence for fields extension: there is a one-to-one, inclusion-reversing correspondence between intermediate subfields of a field extension E/F are in bijection with subgroups of the Galois group $\mathrm{Gal}(E/F)$.

Theorem 1. Let (X, x) be a pointed space that is path-connected, locally path-connected, semi-locally simply connected. Then there is a bijection between the set of isomorphism classes of path-connected pointed coverings and subgroups of $\pi_1(X, x)$. Furthermore, this bijection is an isomorphism of complete lattices. □

We will not prove this result "classically" (but one proof can be found in [Hat02]) as we will express this theorem and prove it in Homotopy Type Theory in section 4.

Let us move on to a short introduction of Homotopy Type Theory.

3. Homotopy Type Theory.

Homotopy Type Theory is an extension of Type Theory that is well-suited for proving continuity-related results. We do not need to prove that some function defined in HoTT is continuous as *any HoTT-defined function is inherently continuous*.

3.1. Basics of Type Theory.

One very important from Type Theory is the ***Curry–Howard Correspondence***: formally proving a result is exactly the same as giving a program whose type is the result. This is what many proof assistants are using under the hood (Rocq, Agda, Lean, *etc*). We specifically chose not to use "proposition" instead of "result" as it has an important meaning in HoTT. In this subsection, we give a succinct overview of Type Theory. We will write $a : A$ to say that a is an expression of type A.

Dependent functions. In Type Theory, we allow the type of $f(x)$ to depend on x. We write the type of f:

$$f : \prod_{x:A} B(x),$$

where A is the type of x and $B(x)$ the type of $f(x)$. We say f is a ***dependent function***. In the case where B doesn't depend on x, then we get back *non*-dependent functions, which we write $f : A \to B$. Under the Curry–Howard correspondence, the Π-type unifies the $\Rightarrow$ and $\forall$–the implication and the universal quantification.

Dependent pairs. Continuing on dependent functions, we define ***dependent pairs***: we allow the type of the 2nd component of the pair to depend on the 1st component. A pair $\langle x, y\rangle$ where $x : A$ and $y : B(x)$ has a type:

$$\langle x, y\rangle : \sum_{x:A} B(x).$$

In the case where B doesn't depend on x, then we get back *non*-dependent pairs, which we write $\langle x, y\rangle : A \times B$. Under the Curry–Howard correspondence, the Σ-type unifies the $\wedge$ and $\exists$–the conjunction and the existential quantification. We will see that, in HoTT, existential quantifications and Σ-types aren't exactly the same.

Universes. What is the type of a type? It's a ***universe*** $\mathcal{U}[\ell]$ for some integer $\ell \in \mathbb{N}$. We call this integer the ***level*** of the universe. The type of $\mathcal{U}[\ell]$ is $\mathcal{U}[\ell + 1]$. This way, we have a hierarchy of universes:

$$\mathcal{U}[0] : \mathcal{U}[1] : \mathcal{U}[2] : \cdots : \mathcal{U}[\ell] : \mathcal{U}[\ell + 1] : \mathcal{U}[\ell + 2] : \cdots .$$

This is a trick to avoid ***Girard's Paradox***: having a type be its own type $\mathcal{U} : \mathcal{U}$ would result in an inconsistent system (we can prove false results). For the remainder of this section, we will write $\mathcal{U}$ "as if" it was the type of all types, leaving the level implicit.

Currying. As often in functional programming, we will identify the types $A \to B \to C$[e] with $A \times B \to C$. As functions can be dependent, this means we will often identify the types:

$$\prod_{a:A} \prod_{b:B(a)} C(a, b) \qquad \text{with} \qquad \prod_{\langle a,b\rangle : \sum_{a:A} B(a)} C(a, b).$$

We will often write the left one for the type, but use the right one for the term, writing $f(x, y)$ instead of $f(x)(y)$ or $f(\langle x, y\rangle)$.

This concludes our short overview of type theory.

3.2. Equality as path.

As we saw in section 2, strict equality is often too restrictive when continuity is involved. What is often called "equality type" in Type Theory has to be reinterpreted to be less restrictive. The answer is ***paths***.

This does makes sense for Algebraic Topology: when comparing paths, we use a homotopy of paths–a path between paths. So "equalities" (which we will now call ***identities*** or ***identifications***, written $\mathsf{Id}_A(x, y)$ or $x =_A y$, and it is a ***type***) of paths in HoTT are homotopies.

Strict equality is also possible, which we will write $\equiv$ (following the convention in [Uni13]), but it is not a type! When x and y are ***strictly*** or ***definitionally equal*** (equal by definition), then we do have a path from x to y; namely, refl_x, the constant path at x.

We do have one important fact to consider, the induction principle of identifications, written $\mathsf{ind}_{=_A}$ in [Uni13][f], but we will call it J to match Cubical Agda's induction principle. It is: for some type A with $a : A$ (often implicit), given a property

$$P : \prod_{b:A} a =_A b \to \mathcal{U},$$

if we have an element u of $P(a, \mathsf{refl}_a)$ then we obtain an element of:

$$\prod_{b:A} \prod_{p:x=_A y} P(b, p).$$

We also require that $\mathrm{J}(P, u, a, \mathsf{refl}_a) \equiv u$. The intuition behind this principle is that: to obtain $P(b, p)$, we move b along p, making p shorter and shorter, until p is refl_a. This induction principle is shown to be consistant with the "equality as path" interpretation.

[e]We write $A \to B \to C$ for $A \to (B \to C)$, as often in functional programming languages.

[f]There are two versions in [Uni13]: based path induction $\mathsf{ind}'_{=_A}$ and path induction $\mathsf{ind}_{=_A}$. They are equivalent (this is shown in [Uni13]). Here, we use based path induction, as it is the one used by Cubical Agda when using `J`.

A beginner mistake is to think that every path should be refl and that, to prove a result about any path, only the refl case need to be considered. This is false: for example, not every loop is refl (we will see later), as to apply J, we need to be able to move one endpoint. This idea that any loop is refl is called ***axiom* K**, which is fundamentally different from J.

$$J : \overbrace{\prod_{A:\mathcal{U}} \prod_{a:A}}^{\text{implicit}} \prod_{P:\prod_{b:B} a=_A b \to \mathcal{U}} P(a, \mathsf{refl}_a) \to \prod_{b:B} \prod_{p:a=_A b} P(b,p)$$

$$K : \underbrace{\prod_{A:\mathcal{U}} \prod_{a:A}}_{\text{implicit}} \prod_{P:a=_A a \to \mathcal{U}} P(\mathsf{refl}_a) \to \prod_{p:a=_A a} P(p).$$

We do not postulate axiom K.[g]

3.3. Transport, action on paths.

The following result tells us that: if we have a path between two points, and some result is true for one point, then it's true for the other point.

Proposition 6 (Transport). If A is a type and $P : A \to \mathcal{U}$ a family of types over A, then we have a function:

$$\mathsf{transport}^P : \overbrace{\prod_{a:A} \prod_{b:B}}^{\text{implicit}} a =_A b \to P(a) \to P(b),$$

such that $\mathsf{transport}^P(\mathsf{refl}_a, x) \equiv x$.

Proof. By path induction on $p : a =_A b$, we consider the case where $b \equiv a$ and $p \equiv \mathsf{refl}_a$. Let $x : P(a)$. We now have to give an element of type $P(a)$. To satisfy the last relation, we give x. □

Another result is that functions are inherently continuous. That is, if there is a path between two points, applying a (non-dependent) function give us a path between the image of two points. After this proposition, we will freely use "map" and "function" when talking about HoTT-defined functions.

Proposition 7 (Application/Action on paths). Given a non-dependent function $f : A \to B$, then we have a function

$$\mathsf{ap}_f : \overbrace{\prod_{a:A} \prod_{b:A}}^{\text{implicit}} a =_A b \to f(a) =_B f(b),$$

such that $\mathsf{ap}_f(\mathsf{refl}_a) \equiv \mathsf{refl}_{f(a)}$.

Proof. By induction on path $p : a =_A b$, we consider the case $b \equiv a$ and $p \equiv \mathsf{refl}$. In this case, we have to give a path from $f(a)$ to $f(a)$–$\mathsf{refl}_{f(a)}$. □

3.4. Types are ∞-groupoids.

The structure we obtain is an ∞-groupoid (see appendix B for more details): we can consider paths $p, q : x =_A y$ between points, paths between paths $r, s : p =_{x=_A y} q$, and so on; and all of these paths are ***invertible*** and ***composable*** and follow basic group-like laws (these are exactly the ones in lemma 1, on page 3).

Proposition 8 (Inverse & composition). Let A be a type and $x, y, z : A$.

- There is a function $\mathsf{inv} : x =_A y \to y =_A x$ such that $\mathsf{inv}(\mathsf{refl}_x) \equiv \mathsf{refl}_x$.
- There is a function $\mathsf{concat} : x =_A y \to y =_A z \to x =_A z$ such that $\mathsf{concat}(\mathsf{refl}_x, \mathsf{refl}_x) \equiv \mathsf{refl}_x$.

We will often write p^{-1} for $\mathsf{inv}(p)$ and $p \cdot q$ for $\mathsf{concat}(p, q)$, matching notations from definition 2 (page 2).

Proof.
- By induction, it suffices to define $\mathsf{inv}(\mathsf{refl}_x) :\equiv \mathsf{refl}_x$.
- By *double* induction,[h] it suffices to define $\mathsf{concat}(\mathsf{refl}_x, \mathsf{refl}_x) :\equiv \mathsf{refl}_x$.

□

Defining these path operations, we want them to follow the same properties as in lemma 1, but we are proving them differently: by path induction. Once every path considered is refl, all of these follow definitionally.

Proposition 9 (Groupoid laws). Let $p : u = v$ and $q : v = w$ and $r : w = x$. We have:

- $(p{\cdot}q){\cdot}r = p{\cdot}(q{\cdot}r)$;
- $p = \mathsf{refl}_x \cdot p$;
- $p^{-1} \cdot p = \mathsf{refl}_y$.
- $p \cdot \mathsf{refl}_y = p$
- $p \cdot p^{-1} = \mathsf{refl}_x$;

□

These laws are known as ***groupoid laws***. A ***groupoid*** is a structure very similar to groups, except that the binary operation · is not defined everywhere. In this case, we can compose paths when the common endpoint matches. There are some conditions on when it is mandatory for the composition to be defined (for example $a \cdot a^{-1}$ and $a^{-1} \cdot a$ must always be defined) in order to have the groupoid structure. In our case, paths follow these conditions. See appendix B for more details on groupoids.

Having this result, we obtain that: ***in HoTT, types are ∞-groupoids*** as these laws are true for paths, paths between paths, paths between paths between paths, *etc*.

3.5. Cubical Agda.

Cubical Agda is the proof assistant used during this internship. It has a Haskell-like syntax where functions are defined in two parts: its type and then its definition.

For example, defining the identity function on some globally defined type `A` is done with:

```
id : A → A
id a = a
```

Notice the use of the Unicode arrow: this is not fancy typesetting but rather a choice of Agda to heavily rely on these symbols in the standard library.

A more robust implementation of the identity function would look like this:

```
id : {ℓ : Level} {A : Type ℓ} → A → A
id a = a
```

making use of implicit arguments `A` and `ℓ` to make the identity function work for any type, at any level in the universe hierarchy (the `Level` type isn't in any universe; it's not a unique instance, as we will see later).

All of what was discussed in subsections 3.1–3.4 is true in Agda, using the correct syntax. Table 1 give a correspondence between Agda's syntax and the one given in subsections 3.1–3.4. Notice that, to define a pair, spaces are important (as variables are only delimited by parentheses and spaces): writing "`(x,y)`" would lead Agda to think you use a variable named "`x,y`".

Agda has different notations regarding definitional/propositional equality from the ones presented before (the chosen conventions here

[g] Agda, by default, does postulate axiom K but it can be disabled. It is always disabled when using Cubical Agda.

[h] There are three ways of defining $p \cdot q$: by induction on p; on q; on both. Depending on our choice, concatenating with refl can be canceled on the left/right by definition. Although, as we will see, canceling of refl_x for concatenation holds propositionally in all cases.

NAME	TYPE THEORY	CUBICAL AGDA
Universe at level ℓ	$\mathcal{U}[\ell]$	`Type ℓ`
Dependent function type	$\prod_{x:A} B(x)$	`(x : A) → B x`
Non-dependent function type	$A \to B$	`A → B`
Dependent pair type	$\sum_{x:A} B(x)$	`Σ A (λ x → B x)`
Non-dependent pair type	$A \times B$	`A × B`
Identity type	$x =_A y$	`x ≡ y`
Function definition	$x \mapsto f(x)$	`λ x → f x`
Pair definition	$\langle x, y\rangle$	`(x , y)`
Path induction	$\mathsf{J}(P, u, p)$	`J P u p`
Action on path p	$\mathsf{ap}_f(p)$	`cong f p`
Transport along p	$\mathsf{transport}^B(p, x)$	`subst B p u`
Path concatenation	$p \cdot q$	`p • q`
Path inversion	p^{-1}	`p ⁻¹`

Table 1 | *Correspondence of syntax*

are from [Uni13]): definitional equality is `=` and not `≡`, and propositional equality (*a.k.a.* identity types) are written `x ≡ y` instead of the usual $x =_A y$.

Also, Cubical Agda is using ***Cubical*** Homotopy Type Theory, and the operations above are defined a bit differently "under the hood".

Paths are special functions. Instead of relying on paths as being some built-in inductive types treated as a black box, in Cubical Agda (as it is defined in Cubical Homotopy Type Theory), paths are, in some special way, functions from `I` to some type `A`: this looks very similar to definition 1 (page 2). This `I` type is, like `Level`, outside the universe hierarchy. It has two special elements: `i0` and `i1`; and 3 operations: minimum written `∧` (infix), maximum written `∨` (infix) and "inversion" $t \mapsto 1 - t$ written `~` (prefix). The only constraint on the definition of a path as a function is that: the endpoints must be definitionally equal to those in the type. This way, we can easily define inv and ap:

```
_⁻¹ : x ≡ y → y ≡ x
p ⁻¹ i = p (~ i)

cong : (f : A → B) → x ≡ y → f x ≡ f y
cong f p i = f (p i)
```

Dependent paths, `PathP`. In HoTT, it is sometimes useful to have a path between two elements of different types, as long as there is a path between them. In [Uni13], transport is used to get back to the "path between two elements of the same type" situation (we will give more details about that in subsection 3.8 when talking about equality in Σ-types). In Cubical Agda, there is an even better way: `PathP`. An expression `p` of type `PathP (λ i → P i) x y` is a *dependent* path between `x : P i0` and `y : P i1` such that `p i : P i` for all `i : I`. The type `x ≡ y` is the special case when `P` doesn't depend on `i`.

Transport, transport everywhere. One of the primitive operations in Cubical HoTT is ***generalized transport***, `transp`. It allows to compute the transport of elements, but also to give a `PathP` between an element and its transport. This is known as `subst-filler`:

```
subst : (B : A → Type ℓ) → x ≡ y → B x → B y
subst B p u = transp (λ i → B (p i)) i0 u

subst-filler : (p : x ≡ y) (B : A → Type ℓ) (u : B x)
    → PathP (λ i → B (p i)) u (subst B p u)
subst-filler p B u i =
    transp (λ j → B (p (i ∧ j))) (~ i) u
```

Understanding why `subst-filler` works and how the special rules for typechecking `transp` are defined is beyond the scope of this document. All you need to remember is that there is a dependent path, `PathP`, from an element to its transport given by `subst-filler`.

In Cubical Agda, transport is everywhere: even `J` is defined in terms of transport. But, unlike in the version of HoTT from [Uni13], we do not have the same definitional equalities:

`subst B refl u ≠ u` and `J P u refl ≠ u`,

but we do have propositional equalities:

```
substRefl u : subst B refl u ≡ u,
 JRefl P u : J P u refl ≡ u.
```

Filling squares, cubes, *etc.* In Cubical Agda, composition of paths is defined a bit differently than in [Uni13]: we use 2D cubes (*a.k.a.* squares) to define them. Cubical Agda give us some primitive operation, `hcomp`, to allow for composition of n-ary composition of m-dimensional cubes. Using `hcomp`, we can define ternary composition (*i.e.* giving the last edge of a square):

$$\begin{tikzcd}
x \arrow[r, dashed, squiggly, "p \bullet\bullet q \bullet\bullet r"] \arrow[d, squiggly, "p"'] & w \\
y \arrow[r, squiggly, "q"'] & z \arrow[u, squiggly, "r"']
\end{tikzcd}$$

```
_••_••_ : x ≡ y
        → y ≡ z
        → z ≡ w
        → x ≡ w
```

Binary composition of paths is defined using this ternary composition but having the first path be `refl`:

$$\begin{tikzcd}
x \arrow[r, dashed, squiggly, "p \bullet q"] \arrow[d, equal, "\mathsf{refl}"'] & z \\
x \arrow[r, squiggly, "p"'] & y \arrow[u, squiggly, "q"']
\end{tikzcd}$$

```
_•_ : x ≡ y → y ≡ z → x ≡ z
p • q = refl •• p •• q
```

Notice that, by using underscores in the name, you can define binary infix operations. You can make some pretty powerful notations this way.

3.6. Equivalences and the Univalence Axiom.

Definition 12. ▹ Two types A and B are ***equivalent***, written $A \simeq B$, when there is a map $f : A \to B$ such that f is an equivalence.

▹ A map $f : A \to B$ is an ***equivalence***, written isEquiv(f), when:

- there exists a map $g : B \to A$ such that for all $b : B$, there exists a path $f(g(b)) =_B b$;
- there exists a map $h : B \to A$ such that for all $a : A$, there exists a path $h(f(x)) =_A x$.

Therefore, as types, we have $A \simeq B :\equiv \sum_{f:A\to B} \mathsf{isEquiv}(f)$ where

$$\mathsf{isEquiv}(f) :\equiv \Big(\sum_{g:B\to A} f \circ g \sim \mathsf{id}_B\Big) \times \Big(\sum_{h:B\to A} h \circ f \sim \mathsf{id}_A\Big),$$

and $u \sim v :\equiv \prod_{x:X} u(x) =_{Y(x)} v(x)$ when $u, v : \prod_{x:X} Y(x)$.

Putting it simply: two types A and B are equivalent when there is a map that is left-invertible and right-invertible. The notation $\sim$ is something seen before: a homotopy of maps. So, equivalence of types is exactly homotopy equivalence of spaces.

Lemma 7. We can define a map

$$\mathsf{idToEquiv} : A =_{\mathcal{U}} B \to A \simeq B,$$

such that $\mathsf{idToEquiv}(\mathsf{refl}_A) :\equiv \mathsf{idEquiv}_A$.

Proof. By induction, we only have to define $\mathsf{idToEquiv}(\mathsf{refl}_A)$. We define it to be the identity equivalence: we define $f, g, h :\equiv \mathsf{id}_A$ and all the paths are refl. □

The ***univalence axiom*** is the following:

Axiom (Univalence axiom)**.** The map idToEquiv itself is an equivalence:

$$(A =_{\mathcal{U}} B) \simeq (A \simeq B).$$

The most important thing to remember is: there is a path between two equivalent types. This map from an equivalence to a path, we will call it

$$\mathsf{ua} : (A \simeq B) \to (A =_{\mathcal{U}} B),$$

(in Cubical Agda, it is also ua).

One more consequence to this univalence axiom is that: there is an induction principle for equivalences (it is known as `JEquiv` in Cubical Agda).

Proposition 10. Given a family of types $P : \prod_{B:\mathcal{U}} A \simeq B \to \mathcal{U}$, if we give an element of type $P(A, \mathsf{idEquiv}_A)$ then we get:

$$\prod_{B:\mathcal{U}} \prod_{e:A\simeq B} P(B, e).$$

□

This univalence axiom allows homotopy type theorists to prove a very simple result that type theorists usually postulate: ***function extensionality***.

Proposition 11 (Function extensionality)**.** Given two dependent functions f and g of type $\prod_{x:A} B(x)$, there is a map

$$\mathsf{funExt}^{-} : f =_{\prod_{x:A} B(x)} g \to \underbrace{\prod_{x:A} f(x) =_{B(x)} g(x)}_{f\sim g}.$$

Assuming the univalence axiom, this map is an equivalence. □

This proof can be found in [Uni13, Section 4.9], but a very simple proof exist in Cubical Homotopy Type Theory:

```
funExt-equiv :
  {A : Type ℓ}
  {B : A → Type ℓ'}
  (f g : (x : A) → B x)
  →
  (f ≡ g) ≃ ((x : A) → f x ≡ g x)
funExt-equiv {A = A} {B = B} f g =
  isoToEquiv (iso funExt⁻ funExt (λ _ → refl) (λ _ → refl))
  where

  funExt⁻ : f ≡ g → (x : A) → f x ≡ g x
  funExt⁻ p x i = p i x

  funExt : ((x : A) → f x ≡ g x) → f ≡ g
  funExt p i x = p x i
```

Here, we define an ***isomorphism*** (same as the equivalence but left- and right-inverses are the same) and then get an equivalence. The important part lies in the definition of `funExt` and `funExt⁻` where we only need to swap `i` and `x` as, in Cubical HoTT, paths look a lot like functions. In this case, no need to use the univalence axiom or do a path induction like we'd have to do in the HoTT proof to define funExt^{-}.

Therefore, $\sim$ and $=$ are equivalent.

3.7. Higher inductive types, HITs.

Higher inductive types look a lot like regular inductive types (using inductive constructors) but allow defining paths. To show the definitions, we will use Cubical Agda code.

Let us start with some simple regular inductive types: booleans, positive integers, the unit type, the empty type; then we'll move on to *higher* inductive types with one important example: the circle (we will see more examples of higher inductive types when explaining truncations in subsection 3.11).

Booleans, 2. Booleans have two constructors that we commonly call true and false (in [Uni13], they are called respectively 1_2 and 0_2). In Cubical Agda, we can define the type of booleans, written $\mathbf{2}$, with the following.

```
data 2 : Type ℓ-zero where
  true  : 2
  false : 2
```

In this code we declare a type named `2`, with two constructors `true` and `false`. This type is at level 0 in the universe hierarchy.

Defined like this, we can do pattern matching on a boolean; it corresponds to case-by-case analysis. This pattern matching allows us to do in Cubical Agda what [Uni13] calls the recursor:

$$\begin{aligned}
\mathsf{rec}_2 &: \prod_{A:\mathcal{U}} A \to A \to \mathbf{2} \to A,\\
\mathsf{rec}_2(A, x, y, \mathsf{true}) &:\equiv x,\\
\mathsf{rec}_2(A, x, y, \mathsf{false}) &:\equiv y;
\end{aligned}$$

it corresponds to the *if then else* construct.

Positive integers, ℕ. We can define positive integers easily with two constructors: zero (for 0) and succ (for $n \mapsto n+1$). The succ constructor takes an argument, another natural number n, such that $\mathsf{succ}(n)$ represents $n+1$.

```
data ℕ : Type ℓ-zero where
  zero : ℕ
  succ : ℕ → ℕ
```

The recursor of the natural numbers allows us to distinguish between zero and succ, but we need to add recursion into the mix to get the induction principle of natural number (this is the usual induction principle very commonly used).

```
indℕ : (P : ℕ → Type ℓ)
  → P zero
  → ((n : ℕ) → P n → P (succ n))
  → (n : ℕ) → P n
indℕ P z s zero     = z
indℕ P z s (succ n) = s n (indℕ P z s n)
```

In Cubical Agda, case by case analysis is done this way: specify definitions for each constructor on separate lines.

Unit type, 1. The unit type $\mathbf{1}$, with one constructor tt, will play an important role when discussing the proof of the Galois correspondence in section 4.

```
data 1 : Type ℓ-zero where
  tt : 1
```

It is terminal in the sense that any function looks like $x \mapsto \mathsf{tt}$ (as we have function extensionality) so, for any type $A : \mathcal{U}$, there is only one function $A \to \mathbf{1}$. Logically, this type corresponds to a tautology (sometimes it can be written $\top$).

Empty type, 0. The empty type $\mathbf{0}$ has no constructor, it is the type equivalent of the empty set $\emptyset$.

```
data 0 : Type ℓ-zero where
```

It is initial in the sense that, for any type $A : \mathcal{U}$, there is only one function $\mathbf{0} \to A$ by simply doing a case-by-case analysis. Logically, this type corresponds to falsity, or some absurdity; it is useful when defining the logical not, as $\neg A :\equiv A \to \mathbf{0}$ (sometimes it can be written $\bot$).

Circle, $\mathbb{S}^1$. The circle has one point base : $\mathbb{S}^1$ and one non-trival identification loop : base $=_{\mathbb{S}^1}$ base. Defining such an identification is possible as we are working with ***higher*** inductive types.

```
data S¹ : Type ℓ-zero where
  base : S¹
  loop : base ≡ base
```

Case-by-case analysis on an element of $\mathbb{S}^1$ is a bit more involved; we need to consider two cases:

1. on the point base,
2. somewhere on the loop loop at position i,

and we must have *continuity*: on the second case, for $i =$ `i0` and $i =$ `i1`, we must get the exact (definitional equality) result we gave for the base case.

As said before, we will study some other higher inductive types when dealing with truncations in subsection 3.11.

3.8. Simplifying identity types.

Sometimes, identity types can be quite complex, but they're often equivalent to simpler types. For example, for booleans $\mathbf{2}$, we define a map:

$$\text{code}_2 : \mathbf{2} \to \mathbf{2} \to \mathcal{U}.$$

by double induction on booleans:

- ▷ $\text{code}_2(\text{true}, \text{true}) :\equiv \mathbf{1}$;
- ▷ $\text{code}_2(\text{true}, \text{false}) :\equiv \mathbf{0}$;
- ▷ $\text{code}_2(\text{false}, \text{true}) :\equiv \mathbf{0}$;
- ▷ $\text{code}_2(\text{fase}, \text{false}) :\equiv \mathbf{1}$.

And, by path induction, we can have that, for all $x, y : \mathbf{2}$, we have a map

$$\text{encode}_2 : x = y \to \text{code}_2(x, y).$$

Then, by case by case analysis on x and y, we can get a map

$$\text{decode}_2 : \text{code}_2(x, y) \to x = y.$$

We can then get that encode$_2$, is an equivalence by showing that decode$_2$ is a left- and right-inverse (one by case-by-case analysis, the other by path induction).

This general pattern (code, encode, decode and show that they are inverses) is very important, and can be applied to many scenarii ($\mathbb{N}$, disjoint union, $\mathbb{S}^1$, *etc*).

Another simplification of identity types is for Σ-types. One might suspect that there is an equivalence:

$$\big(\langle a, b\rangle = \langle a', b'\rangle\big) \simeq (a = a') \times (b = b'),$$

but this is ill-typed for dependent pairs (for non-dependent pairs, this is true): b has type $B(a)$ but b' has type $B(a')$. We need to transport b along the equality $a = a'$, obtaining:

$$\big(\langle a, b\rangle =_{\sum_{a:A} B(a)} \langle a', b'\rangle\big) \simeq \sum_{p:a=_A a'} \text{transport}^B(p, b) =_{B(a')} b'.$$

In Cubical Agda, instead of using transport^B, we can also provide a `PathP` between b and b'.

3.9. An illustrative example.

In this subsection, we will see how we can define a path

$$(\mathbf{2}, \text{true}) = (\mathbf{2}, \text{false}).$$

If we want this path in the space $\mathcal{U} \times \mathbf{2}$, then no luck ! However, in the space $\sum_{A:\mathcal{U}} A$, we can provide such a path.

Firstly, let us define the non-trivial automorphism of $\mathbf{2}$, not:

$$\begin{aligned} &\text{not} : \mathbf{2} \to \mathbf{2} \\ &\text{not}(\text{true}) :\equiv \text{false} \\ &\text{not}(\text{false}) :\equiv \text{true}. \end{aligned}$$

This map not induces an equivalence $\mathbf{2} \simeq \mathbf{2}$. Then, by univalence, we obtain a non-trivial path p (the trivial automorphism, identity, would have given us the path refl) between $\mathbf{2}$ and $\mathbf{2}$.

After, that, we need to see that

$$\text{transport}(p, \text{true}) =_2 \text{false},$$

which is given by the other elements of the equivalence in the univalence axiom (see [Uni13, Remark 2.10.4]). This is known as the "propositional computation rule" of the univalence axiom: transport along some ua-equivalence-induced path correspond (propositional equality, not definitional) to applying the function in the equivalence.

Therefore, we obtain a path

$$(\mathbf{2}, \text{true}) =_{\sum_{A:\mathcal{U}} A} (\mathbf{2}, \text{false}).$$

3.10. *n*-types.

Let us finally explain the *real* meaning of a *proposition* in HoTT and what *sets* are.

Definition 13. ▹ A ***contractible*** type, or ***(−2)-type***, is a type A such that there exists a point $a : A$ such that there is a map $\prod_{a':A} a =_A a'$.

- ▹ A ***(mere) proposition***, or ***(−1)-type***, is a type A such that, for any elements $a, a' : A$, $a = a'$.
- ▹ A ***set***, or ***0-type***, is a type A such that, for any elements $a, a' : A$, the type $a =_A a'$ is a proposition.
- ▹ A ***(1-)groupoid***, or ***1-type***, is a type A such that, for any $a, a' : A$, the type $a =_A a'$ is a set.
- ▹ A ***2-groupoid***, or ***2-type***, is a type A such that, for any $a, a' : A$, the type $a =_A a'$ is a groupoid.
- ▹ A ***n-type*** for $n \geq -1$ is a type A such that, for any elements $a, a' : A$, the type $a =_A a'$ is a $(n-1)$-type.

Intuitively, a type is contractible when it is equivalent to $\mathbf{1}$. Given such a definition, we could think that the circle is contractible: we can always provide a path between two points on the circle, as it is path-connected. However, our choice for the path must be continuous (induction principle of the circle), and we can't provide a continuous choice, thus the circle isn't contractible.

A mere proposition is a type such that there is at most one proof (up to path), so it is either equivalent to $\mathbf{0}$ or $\mathbf{1}$. For example, the isEquiv(f) type from definition 12 is a proposition. The type is-n-type(A) of proofs that A is an n-type (which you can easily define by induction on n) is a proposition.

SPACE	CONTRACTIBLE	PROPOSITION	SET	GROUPOID	2-GROUPOID
0	✗	✓	✓	✓	✓
1	✓	✓	✓	✓	✓
2	✗	✗	✓	✓	✓
$\mathbb{N}$	✗	✗	✓	✓	✓
$\mathbb{S}^1$	✗	✗	✗	✓	✓

Table 2 | *Some n-types*

In a set, axiom K (from subsection 3.2, page 6) holds: any loop is refl. A set is *discrete* as there are, up to homotopy, at most one path between two points.

In a groupoid, you are allowed to have non-trivial loops. For example, the circle $\mathbb{S}^1$ is a groupoid. We will discuss a bit more about groupoids in section 4 when explaining deloopings of groups.

Lemma 8. If a type is an n-type then it is also an $(n+k)$-type for any positive integer $k : \mathbb{N}$. □

Table 2 gives examples of types (those defined previously) and if they are n-types for $n \in \{-2, -1, 0, 1\}$.

3.11. Truncations.

Sometimes, it can be useful to turn some type into an n-type. ***Truncation*** is the universal way of doing this transformation.

Definition 14. The ***n-truncation*** of a type A, written $\|A\|_n$, is defined by the following properties:

- there is an inclusion map $|-|_n : A \to \|A\|_n$;
- it is an n-type.[i]

This is, in fact, a higher inductive definition. Sometimes, it can be useful to "extract" an element $a : A$ in $\|A\|_n$: this is known as ***truncation elimination***. You call only perform elimination of an n-truncation when eliminating to an n-type.

Proposition 12 (Truncation elimination). Given an n-type B, a map $f : A \to B$ induces a map $\tilde{f} : \|A\|_n \to B$ such that

$$\begin{tikzcd} A \arrow[r, "|-|_n"] \arrow[dr, "f"'] & \|A\|_n \arrow[d, "\tilde{f}"] \\ & B \end{tikzcd}$$

commutes. □

In [Uni13], a different construction (the "hubs and spokes" construction) is given for the n-truncation but, for most things, the implementation details don't matter too much.

Proposition 13. The operation $\|-\|_n$ is functorial: a map $f : A \to B$ induces a map $\|f\|_n : \|A\|_n \to \|B\|_n$ such that the following square commutes:

$$\begin{tikzcd} A \arrow[r, "|-|_n"] \arrow[d, "f"'] & \|A\|_n \arrow[d, "\|f\|_n"] \\ B \arrow[r, "|-|_n"'] & \|B\|_n, \end{tikzcd}$$

i.e. $\|f\|_n(|a|_n) = |f(a)|_n$.

Proof. By elimination (as $\|B\|_n$ is an n-type), the map $|-|_n \circ f : A \to \|B\|_n$ induces a map $\|f\|_n : \|A\|_n \to \|B\|_n$. □

[i]We are allowed this in the HIT definition as is-n-type($\|A\|_n$) is $(2n+4)$-variable map that results in a path.

Proposition 14. A type A is an n-type if and only if $|-|_n : A \to \|A\|_n$ is an equivalence thus, by the univalence axiom, if and only if $\|A\|_n = A$. □

Propositional truncation. Let us define "being path-connected" as a type in HoTT. The usual definition is: for any two points, there is a path between those two points; so, we'd be tempted to define

$$\mathsf{isConnected}(A) :\cancel{\equiv} \prod_{a,a':A} a = a'.$$

But this is just the definition of being a proposition, something isn't right. What we need is to give proof that such a path exists, without explicitly giving the path. This is where propositional truncation, or (-1)-truncation, comes in: with the inclusion map, we can provide such a path, but we can't access it unless we are eliminating to a proposition. We therefore define:

$$\mathsf{isConnected}(A) :\equiv \prod_{a,a':A} \|a = a'\|_{-1}.$$

Usually, when writing "there (***merely***) exists $x : A$ such that $P(x)$", we translate it to: $\left\| \sum_{x:A} P(x) \right\|_{-1}$.

Paths in n-truncations. When dealing with paths in n-truncations, there is an important equivalence of types (we'll be using it in section 4). It can be proven using the encode/decode method (see [Uni13, Theorem 7.3.12]).

Proposition 15. For any $x, y : A$, there is an equivalence:

$$\|x =_A y\|_n \simeq \left(|x|_{n+1} =_{\|A\|_{n+1}} |y|_{n+1}\right).$$

□

This concludes our presentation of Homotopy Type Theory and Cubical Agda. We can now move on to the proof of the Galois Correspondence in HoTT and its formalization in Cubical Agda.

4. The Galois Correspondence in HoTT.

To properly understand the proof, we start by introducing some useful types, notations, and lemmas.

4.1. Homotopy pullbacks.

Given two maps $f : A \to C$ and $g : B \to C$, we can take its *homotopy pullback* $A \times_C B$ such that this diagram commutes:

$$\begin{tikzcd} A \times_C B \arrow[r, "\mathsf{pr}_1"] \arrow[d, "\mathsf{pr}_2"'] \arrow[dr, phantom, "\lrcorner", very near start] & A \arrow[d, "f"] \\ B \arrow[r, "g"'] & C. \end{tikzcd}$$

It corresponds to the type:

$$A \times_C B :\equiv \sum_{a:A} \sum_{b:B} f(a) = g(b).$$

For the purpose of this internship, some properties on pullbacks where proven (*see* `Pullback.agda` *for all Cubical Agda proofs related to pullbacks*). Appendix B gives more details on pullbacks.

Lemma 9 (Commutativity). We have $A \times_C B = B \times_C A$. □

The *pasting lemma* is a common lemma used in Category Theory. We give two versions of it: one vertical and one horizontal.

Lemma 10 (Horizontal pasting lemma). We have $(A \times_D C) \times_C B = A \times_D B$. This correspond to the equality between the left-square pullback and the full-rectangle pullback:

$$\begin{tikzcd}
X \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & A \times_D C \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & A \arrow[d] \\
B \arrow[r] & C \arrow[r] & D.
\end{tikzcd}$$

□

Lemma 11 (Vertical pasting lemma)**.** We have $B \times_C (C \times_D A) = B \times_D A$. This correspond to the equality between the upper-square pullback and the full-rectangle pullback:

$$\begin{tikzcd}
X \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & B \arrow[d] \\
C \times_D A \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & C \arrow[d] \\
A \arrow[r] & D.
\end{tikzcd}$$

□

Definition 15. Given a map $f : A \to B$ and $b : B$, the ***(homotopy) fiber*** of f at b is

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

The connection between homotopy fibers and homotopy pullbacks is very simple and yet very useful as we'll see:

$$\begin{tikzcd}
\mathsf{fib}_f(b) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{1} \arrow[d, "\mathsf{pick}_b"] \\
A \arrow[r, "f"'] & B
\end{tikzcd}
\qquad
\begin{aligned}
A \times_B \mathbf{1} &\equiv \sum_{a:A} \sum_{u:\mathbf{1}} f(a) = \mathsf{pick}_b(u) \\
&= \sum_{a:A} f(a) = b \\
&\equiv \mathsf{fib}_f(b).
\end{aligned}$$

where $\mathsf{pick}_b = u \mapsto b$.

4.2. Deloopings of groups.

A group G can be represented in different ways in HoTT: one simple idea is as an element of the type

$$\sum_{G:\mathcal{U}} \sum_{f:G\to G\to G} \sum_{i:G\to G} \sum_{z:G} \mathsf{is\text{-}assoc}(f) \times \mathsf{is\text{-}inverse}(f,i,z) \times \mathsf{is\text{-}neutral}(f,z),$$

where is-assoc, is-inverse and is-neutral are defined as expected. This is known as an ***external*** representation. A simpler (and equivalent) way of seeing is using a ***delooping*** of a group, this is known as an ***internal*** representation of a group.

Definition 16. The ***loop space*** of a pointed type (A,a) is the type $\Omega(A,a) :\equiv a =_A a$. The ***fundamental group*** of a pointed type (A,a) is the type $\pi_1(A,a) :\equiv \|a =_A a\|_0$, *i.e.* the set of loops at a up to homotopy.

When the type is a groupoid, then its loop space is the fundamental group (because $a =_A a$ is a set). It has a group-like structure with concatenation of paths and path-reversing as operations and refl_a as neutral element.

One very simple representation of a group G would be as a pointed groupoid (A,a) such that $\pi_1(A,a)$ is the group G. Does a groupoid like this always exist? The answer is yes, it is known as a ***delooping*** of G, written $\mathbf{B}\,G$, and it can be defined as a higher inductive type.

Proposition 16 ([CMO25, Theorem 3])**.** For any group G, there is a pointed connected groupoid $\mathbf{B}\,G$ such that $\Omega(\mathbf{B}\,G) = G$. Conversely, for any pointed connected groupoid (A,a), we have that $\mathbf{B}\,\Omega(A,a) = (A,a)$. These operations Ω and $\mathbf{B}$ are functorial. We therefore have an equivalence of categories:

$$\begin{tikzcd}
\textbf{Pointed Connected Groupoids} \arrow[r, bend left, "\Omega"] & \textbf{Groups} \arrow[l, bend left, "\mathbf{B}"]
\end{tikzcd}.$$

In [CMO25], two constructions for deloopings are developed in Cubical Agda. One of the constructions as a higher inductive type:

- ▷ define a basepoint $\star_{\mathbf{B}\,G}$;
- ▷ define a non-trivial loop $\ell_g : \star_{\mathbf{B}\,G} = \star_{\mathbf{B}\,G}$ for every element g;
- ▷ define an "identification of loops" (a homotopy) $\ell_g \cdot \ell_h = \ell_{gh}$ for every $g, h \in G$;
- ▷ define it to be a groupoid.

In appendix B, a category-theory-based definition of this delooping is given.

For the remainder of this document, we will be using deloopings of groups. The fundamental group of a (pointed connected) type has a simple delooping: its 1-truncation.

Proposition 17 ([MO25, Proposition 30])**.** For a pointed connected type (A,a), one delooping of its fundamental group $\pi_1(A,a)$ is $\|A\|_1$.

The proof of this proposition uses the results on truncations and, in particular, proposition 15:

$$\Omega(\|A\|_1, |a|_1) \equiv \left(|a|_1 =_{\|A\|_1} |a|_1\right) \overset{\text{prop. 15}}{=} \|a =_A a\|_0 \equiv \pi_1(A,a).$$

As the Galois correspondence deals with *subgroups* of $\pi_1(A,a)$, we have to represent subgroups with the deloopings.

Definition 17. We say H is a ***subgroup*** of G when there is an injective group homomorphism $i : H \hookrightarrow G$.

In the formal proof (in Cubical Agda), we do not want to have to deal with deloopings directly, just with the pointed connected groupoids. So, it is important to characterize the delooping of an injection.

Lemma 12 ([Uni13, Theorem 7.2.1])**.** Given a type A, its loop spaces $\Omega(A,a)$ are contractible for every a if and only if A is a set. □

Proposition 18. We have $i : H \hookrightarrow G$ is injective if and only if the fibers of $\mathbf{B}\,i : \mathbf{B}\,G \to \mathbf{B}\,H$ are sets (this is known as a 0-truncated map).

Proof. By [MO24, Lemma 11], we have that $\mathbf{B}(\ker i) = \ker(\mathbf{B}\,i)$ where $\ker(\mathbf{B}\,i) :\equiv \mathsf{fib}_{\mathbf{B}\,i}(\star_{\mathbf{B}\,H})$ where $\star_{\mathbf{B}\,H}$ is the basepoint of $\mathbf{B}\,H$.

- ▹ Suppose i injective and let $x : \mathbf{B}\,H$. As $\mathbf{B}\,H$ is connected and being a set is a proposition, we have a path from $\star_{\mathbf{B}\,H}$ to x. So, we only need to consider the case where $x = \star_{\mathbf{B}\,H}$ (we can transport the result along the previous path). We have that:

  $$\Omega\ker(\mathbf{B}\,i) = \Omega\,\mathbf{B}(\ker i) = \ker i,$$

  which is trivial (contractible/equivalent to $\mathbf{1}$), thus $\ker(\mathbf{B}\,i)$ is a set.
- ▹ Suppose $\mathbf{B}\,i$ is 0-truncated. We have:

  $$\ker i = \Omega\,\mathbf{B}(\ker i) = \Omega\underbrace{\ker(\mathbf{B}\,i)}_{\text{set}},$$

  therefore $\ker i$ is trivial (contractible/equivalent to $\mathbf{1}$), thus i is injective.

□

4.3. Some useful types.

For the remainder of this section, fix some pointed path-connected type (A,a). We define the type of pointed path-connected covering spaces of (A, a) to be:

$$\mathsf{Covering}(A, a) :\equiv \sum_{B:\mathcal{U}} \sum_{b:B} \sum_{p:B\to A} \begin{pmatrix} \mathsf{isCovering}(p) \\ \times \\ \mathsf{isConnected}(B) \\ \times \\ p(b) = a \end{pmatrix},$$

where $\mathsf{isCovering}(p) :\equiv \prod_{a':A} \mathsf{isSet}(\mathsf{fib}_p(a))$.

Let us justify why such a definition is acceptable. The arguments come from [HH18]:

> *The use of neighborhoods can be largely avoided because every space constructed by the standard geometric realization of a simplicial set is a CW-complex and thus satisfies all local connectedness properties (for example local path-connectedness or semi-local simple connectedness). Moreover, every construct in type theory is continuous under this interpretation. Therefore, there is no need to mention local connectivity or continuity, because we cannot define any "bad" space in the type theory.*
>
> *Homeomorphism is weakened to homotopic equivalence because, again, it is impossible to distinguish homeomorphic but not homotopic objects inside the type theory.*

In a way, HoTT only allows us to define well-behaved spaces and operations so we can take a step back from the usual topological details (local path-connectedness for example) and focus on the important matter.

Next, we define the type of subgroups. Intuitively (without considering deloopings), this type would be $\sum_{H:\mathsf{Group}} H \hookrightarrow G$. But, using deloopings and thanks to proposition 18, we obtain an easier-to-deal-with type:

$$\mathsf{Subgroup}(G) :\equiv \sum_{\mathsf{B}H:\mathcal{U}} \sum_{\star_{\mathsf{B}H}:\mathsf{B}H} \sum_{\mathsf{B}i:\mathsf{B}H\to\mathsf{B}G} \begin{pmatrix} \mathsf{isConnected}(\mathsf{B}H) \\ \times \\ \mathsf{isGroupoid}(\mathsf{B}H) \\ \times \\ \mathsf{is\text{-}0\text{-}truncated}(\mathsf{B}i) \\ \times \\ \mathsf{B}i(\star_{\mathsf{B}H}) = \star_{\mathsf{B}G} \end{pmatrix}.$$

Notice the use of $\mathsf{B}G$ instead of $\mathbf{B}\,G$: here, we don't manipulate deloopings directly, but rather use them as pointed connected groupoids such that the inclusion map is pointed and 0-truncated. For the special case where $G \equiv \pi_1(A, a)$, we use:

$$\mathsf{Subgroup}(\pi_1(A, a)) :\equiv \sum_{\mathsf{B}H:\mathcal{U}} \sum_{\star_{\mathsf{B}H}:\mathsf{B}H} \sum_{\mathsf{B}i:\mathsf{B}H\to\|A\|_1} \begin{pmatrix} \mathsf{isConnected}(\mathsf{B}H) \\ \times \\ \mathsf{isGroupoid}(\mathsf{B}H) \\ \times \\ \mathsf{is\text{-}0\text{-}truncated}(\mathsf{B}i) \\ \times \\ \mathsf{B}i(\star_{\mathsf{B}H}) = |a|_1 \end{pmatrix}.$$

The goal of the proof of the Galois Correspondence is to show that there is an equivalence

$$\mathsf{Subgroup}(\pi_1(A, a)) \simeq \mathsf{Covering}(A, a).$$

For the rest of the section, we will write $\mathbf{B}\,G$, $\mathbf{B}\,i$, *etc* but treat it as if we are not manipulating a delooping directly.

The next subsection will describe a construction of the universal cover. After that, we will prove the Galois correspondance (by giving an isomorphism) in 4 parts:

SECTION 4.5. given a subgroup, give a pointed connected covering;

SECTION 4.6. given a covering, give a subgroup of $\pi_1(A, a)$;

SECTION 4.7. given a subgroup, getting its pointed connected covering and then getting the subgroup of this covering leads to the same group;

SECTION 4.8. given a covering, getting its subgroup and then getting the covering of this subgroup leads to the same covering.

We will use the original proof in [MO25] as a basis, but the details can be a bit different as the goal was to implement this proof in Cubical Agda (without the use of explicit deloopings of groups).

4.4. Universal cover.

One way to construct the universal cover is with homotopy classes of paths starting at some point, as seen in proposition 4. In HoTT, this construction is translated into the type

$$\sum_{b:A} \|a =_A b\|_0,$$

and, as said before, the assumptions that the space is "well-behaved" so we do not need to assume A to be locally path-connected or semi-locally simply-connected.

Proposition 19 ([HH18, Lemma 10]). For some path-connected pointed type (A, a), the type

$$\mathsf{univ}(A, a) :\equiv \sum_{b:A} \|a =_A b\|_0$$

is a simply-connected covering of A, thus is the universal cover of A.

Proof. See `UniversalCovering.agda`. □

To understand how simply-connectedness is defined in HoTT, it is easier to go back to the definition of connectedness. An equivalent definition of $\mathsf{isConnected}(A)$ for some non-empty type A is: its set-truncation $\|A\|_0$ is contractible. Set truncation removes any "holes" (such as the one in $\mathbb{S}^1$) and is thus equivalent to the set of connected components. Simply-connecteness of a type A can now be defined by: its 1-truncation $\|A\|_1$ is contractible. This way, we keep 2D "holes" but fill higher dimensional ones (*e.g.* the one in $\mathbb{S}^2$, the 3D sphere) and, this way, $\|A\|_1$ is contractible if and only if there are no 2D "holes".

One important fact is that the universal cover is the fiber of the inclusion map $|{-}|_1$:

$$\mathsf{univ}(A, a) \;\equiv\; \sum_{b:A} \|a =_A b\|_0 \;=\; \sum_{b:A} \left(|a|_1 =_{\|A\|_1} |b|_1\right) \;=\; \mathsf{fib}_{|-|_1}(|a|_1).$$

Thus, it is the pullback:

$$\begin{tikzcd} \mathsf{univ}(A, a) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{1} \arrow[d, "\mathsf{pick}_{|a|_1}"] \\ A \arrow[r, "|-|_1"] & \|A\|_1 \end{tikzcd}.$$

4.5. From Subgroups to Coverings.

See `SubgroupToCovering.agda` *for this part of the proof in Cubical Agda.*

Fix a subgroup $G \leqslant \pi_1(A, a)$ with $i : G \hookrightarrow \pi_1(A, a)$ its inclusion map (as said before, we will only consider its delooping $\mathbf{B}\,G$ and the delooping of its inclusion map $\mathbf{B}\,i$). We construct the covering C_G as the homotopy pullback:

$$\begin{tikzcd} C_G \arrow[r] \arrow[d, "p_G"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{B}\,G \arrow[d, "\mathbf{B}\,i"] \\ A \arrow[r, "|-|_1"'] & \|A\|_1 \end{tikzcd}.$$

The projection map $p_G : C_G \to A$ will be the covering map. To show that its fiber over an arbitrary element a' is a set, we consider the following commutative pullback:

$$\begin{tikzcd}
F(a') \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & C_G \arrow[r] \arrow[d, "p_G"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{B}\,G \arrow[d, "\mathbf{B}\,i"] \\
\mathbf{1} \arrow[r, "\mathsf{pick}_{a'}"'] \arrow[rr, bend right, "\mathsf{pick}_{|a'|_1}"'] & A \arrow[r, "|{-}|_1"'] & \|A\|_1
\end{tikzcd} .$$

By the pasting lemma, we have the equality of the two pullbacks:

$$\mathsf{fib}_{p_G}(a') = \mathbf{1} \times_A C_G = \mathbf{1} \times_{\|A\|_1} \mathbf{B}\,G = \mathsf{fib}_{\mathbf{B}\,i}(|a'|_1),$$

which, by proposition 18 is a set.

Next, we define the point

$$\star_{C_G} :\equiv (\star_{\mathbf{B}\,G}, a, r) : C_G,$$

where $r : \mathbf{B}\,i(\star_{\mathbf{B}\,G}) = |a|_1$ is the “pointing equality” of $\mathbf{B}\,i$. We thus have *definitionally* that $p_G(\star_{C_G}) = a$ as p_G is the projection map along the A-component.

Finally, we have to show that C_G is path-connected. To do that, we consider the following commutative diagram:

$$\forall g : \mathbf{B}\,G \qquad \begin{tikzcd}
\mathsf{univ}(A,\tilde{a}) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{1} \arrow[d, "\mathsf{pick}_g"] \\
C_G \arrow[r, "u"] \arrow[d, "p_G"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{B}\,G \arrow[d, "\mathbf{B}\,i"] \\
A \arrow[r, "|{-}|_1"'] & \|A\|_1
\end{tikzcd} .$$

We have that:

$$\begin{aligned}
C_G &= \sum_{g:\mathbf{B}\,G} \mathsf{fib}_u(g) && \text{by [Uni13, Lemma 4.8.2] and ua} \\
&= \sum_{g:\mathbf{B}\,G} \mathsf{univ}(A,\tilde{a}) && \text{by vertical pasting lemma,}
\end{aligned}$$

where $\tilde{a}$ is the 1-truncation-elimination of $\mathbf{B}\,i(g)$ (which is possible as being connected is a proposition, so it is also a groupoid). And, we have that $\mathbf{B}\,G$ is connected and $\mathsf{univ}(A,\tilde{a})$ is connected, thus C_G is path-connected.

4.6. From Coverings to Subgroups.

See `CoveringToSubgroup.agda` for this part of the proof in Cubical Agda.

Consider a pointed path-connected covering (X, x) with the map $p : X \to A$ such that p is 0-truncated (its fibers are sets). The usual way of mapping a covering to a subgroup of $\pi_1(A,a)$ is by considering the fundamental group of the covering $\pi_1(X,x)$. Here, as we are considering a *connected* covering, its 1-truncation $\|X\|_1$ is a delooping $\mathbf{B}\,\pi_1(X,x)$.

By proposition 13, we have

$$\begin{tikzcd}
X \arrow[r, "|{-}|_1"] \arrow[d, "p"'] & \|X\|_1 \arrow[d, "\|p\|_1"] \\
A \arrow[r, "|{-}|_1"'] & \|A\|_1
\end{tikzcd}$$

is a commutative diagram.

The type $\|X\|_1$ is a groupoid, it is connected as connectedness is preserved by truncation ($\|\|X\|_1\|_0 = \|X\|_0$ which is contractible). As the given covering is pointed, we have $r : p(x) = a$, thus by

$$\|p\|_1(|x|_1) = |p(x)|_1 \overset{\mathsf{ap}_{|-|_1}(r)}{=} |a|_1,$$

we conclude that $\|p\|_1$ is a pointed map.

We only need to prove one last thing: the 1-truncation $\|p\|_1$ of the 0-truncated map p is still 0-truncated. To do that we prove the following lemma.

Lemma 13. For any $a : A$, there is a path $\mathsf{fib}_p(a) = \mathsf{fib}_{\|p\|_1}(|a|_1)$.

Proof. Using ua, we only need an isomorphism between the two types.

- ▷ Given $x : X$ and $r : p(x) =_A a$ (deconstructing the element of the fiber $\mathsf{fib}_p(a)$), then we give $|x|_1, \mathsf{transport}(q, r^{-1})$ as the path where r is the path given by proposition 15.
- ▷ Given $u : \|X\|_1$ and $q : \|p\|_1(u) =_{\|A\|_1} |a|_1$, we can eliminate u as $\mathsf{fib}_p(a)$ is a set, and we get $x : X$ and $q' : |p(x)|_1 = |a|_1$. We can also eliminate $\mathsf{transport}(q', r) : \|p(x) = a\|_1$. Finally we construct the element of $\mathsf{fib}_p(x)$.
- ▷ The rest of the proof (proof that these two operations are inverses of each other) can be found in `CoveringToSubgroup.agda`.

□

Lemma 14. The map $\|p\|_1$ is 1-truncated, *i.e.* for any element $v : \|A\|_1$, the fiber $\mathsf{fib}_{\|p\|_1}(v)$ is a set.

Proof. As being a set is a proposition (thus a groupoid), we do truncation elimination on v, and apply the previous lemma. □

4.7. Left inverse.

See `LeftInv.agda` and the folder `LeftInv/` for this part of the proof in Cubical Agda.

The Agda proof is more technical than the one presented here or the one presented in [MO25]. Feel free to look at the Agda code for more details.

We consider $G \leqslant \pi_1(A,a)$ with its inclusion map $i : G \hookrightarrow \pi_1(A,a)$. Apply subsection 4.5 (subgroup to covering) and then subsection 4.6 (covering to subgroup) leads to the following diagram:

$$\begin{tikzcd}
C_G \arrow[r] \arrow[d, "p_G"'] \arrow[dr, phantom, "\lrcorner", very near start] \arrow[rr, bend left, "|{-}|_1"] & \mathbf{B}\,G \arrow[d, "\mathbf{B}\,i"] & \|C_G\|_1 \arrow[dl, "\|p_G\|_1"] \\
A \arrow[r, "|{-}|_1"'] & \|A\|_1 &
\end{tikzcd} .$$

We will write $q^\star : \mathbf{B}\,i(\star_{\mathbf{B}\,G}) = a$.

We want to give

- ▷ a path from $\|C_G\|_1$ to $\mathbf{B}\,G$;
- ▷ a dependent path from $\|p_G\|_1$ to $\mathbf{B}\,i$;
- ▷ a dependent path from $\star_{\mathbf{B}\,G}$ to $|(a, \star_{\mathbf{B}\,G}, q^\star)|_1$;
- ▷ a dependent path from $q^\star$ to $\mathsf{refl}_{|a|_1}$.

There is no need to provide a path from the proof that $\mathbf{B}\,G$ is connected to the one that $\|C_G\|_1$ is connected (and same for isGroupoid) as it is a proposition (and all elements of a proposition are equal).

The hardest part is the last one, not mentioned in [MO25], but we will give a short presentation of how the Agda proof works and refer the interested reader to the code for more technical details.

Let us start with the path from $\mathbf{B}\,G$ to $\|C_G\|_1$. We use the univalence axiom and define an isomorphism between the two types.

- ▷ Given an element of $\|C_G\|_1$ we construct an element of $\mathbf{B}\,G$ by eliminating (as $\mathbf{B}\,G$ is a groupoid), thus getting $(a, g, r) : C_G$: we use the element $g : \mathbf{B}\,G$.
- ▷ Given an element g of $\mathbf{B}\,G$, we eliminate $\mathbf{B}\,i(g) : \|A\|_1$ thus getting an element $a : A$ with $r : \mathbf{B}\,i(g) = |a|_1$, and thus we use the element $|(a, g, r)|_1 : \|C_G\|_1$.
- ▷ Those two operations are inverses of each other (see the code for the full proof).

$$\begin{tikzcd} \mathbf{B}\,G \arrow[r, bend left, "t"] \arrow[r, equal, "q"] & \|C_G\|_1 \arrow[l, bend left, "s"] \end{tikzcd} .$$

Then, we have the following concatenation of paths:

$$\begin{aligned} \mathbf{B}\,i &\overset{dependent}{\rightsquigarrow} x \mapsto \mathsf{transport}(\mathsf{refl}, \mathbf{B}\,i(\mathsf{transport}(q, x))) \\ &= \mathbf{B}\,i \circ s \\ &= \|p_G\|_1. \end{aligned}$$

The first dependent path is a lemma called `funTypeTransp` in Cubical Agda: it tells us that the transport of a function corresponds to pre- and post-composing with the respective transports. The second path is non-dependent and corresponds to the "propositional computation rule of ua" (*c.f.* subsection 3.9). The last one is also non-dependent and it simply (function extensionality) is a truncation elimination on the argument of type $\|C_G\|_1$ and consider the 3rd component of that element, it is of type $\mathbf{B}\,i(g) = |a|_1$.

After that, we have a dependent path $\star_{\mathbf{B}G} \rightsquigarrow_{dependent} |(a, \star_{\mathbf{B}G}, q^\star)|_1$ by simply transporting on the equality q and apply the propositional computation rule of univalence.

Finally we have to give a dependent path between $q^\star$ and $\mathsf{refl}_{|a|_1}$ corresponding to the equality of the two "pointing equalities." This part is a lot more developed in the Agda code but, to put it simply, we have to give a non-dependent path between:

$$\mathsf{transport}_2^{f,x \mapsto f(x)=|a|_1}(q_1, q_2, q^\star) = \mathsf{refl}_{|a|_1},$$

where q_1 and q_2 are respectively the path between $\mathbf{B}\,i$ and $\|p_G\|_1$, and between $\star_{\mathbf{B}G}$ and $|\star_{C_G}|_1$, and $\mathsf{transport}_2$ is the 2-variable version of transport. By [Uni13, Theorem 2.11.3], and with Agda-like notations, it is equivalent to giving a path $(\lambda j.\ q_1\ j\ (q_2\ j)) = q^\star$. We can then decompose q_1 and q_2 in three part each and, as this function-path-application (named `congP` in Cubical Agda) works well with path decompositions. We can thus look at each part separately. Each part is located in its own file: `LeftInv/Part1.agda`, `LeftInv/Part2.agda` and `LeftInv/Part3.agda`.

4.8. Right inverse.

See `RightInv.agda` and the folder `RightInv/` for this part of the proof in Cubical Agda.

Fix some path-connected pointed covering (X, x) with the covering map $p : X \to A$ and where $q^\star : p(x) = a$. Apply subsection 4.6 (covering to subgroup) and then subsection 4.5 (subgroup to covering) leads to the following diagram:

$$\begin{tikzcd} X \arrow[r, dashed, "e"] \arrow[rr, bend left, "|-|_1"] \arrow[dr, "p"'] & C_{\pi_1(X)} \arrow[r] \arrow[d, "\tilde{p}"] \arrow[dr, phantom, "\lrcorner", very near start] & \|X\|_1 \arrow[d, "\|p\|_1"] \\ & A \arrow[r, "|-|_1"] & \|A\|_1 \end{tikzcd} ,$$

where $C_{\pi_1(X)} \equiv \sum_{a:A} \sum_{u:\|X\|_1} \|p\|_1(u) = a$ and the e map is obtained by:

$$e(x) :\equiv \left(p(x), |x|, \mathsf{refl}_{|p(x)|_1}\right) : C_{\pi_1(X)}.$$

An inverse map $e' : C_{\pi_1(X)} \to X$ is possible by truncation-elimination on the $\|X\|_1$-component, but it requires eliminating to a groupoid. The trick is to use fibers: we define a map $\mathsf{fib}\text{-}e' : \prod_{a:A} \mathsf{fib}_{\tilde{p}}(a) \to \mathsf{fib}_p(a)$ and we can eliminate here as $\mathsf{fib}_p(a)$ is a set. To define $\mathsf{fib}\text{-}e'(a, ((u, a', q), r))$ where

$$\begin{aligned} a, a' &: A, \\ u &: \|X\|_1, \\ q &: \|p\|_1(u) = |a'|_1 \\ r &: a' = a, \end{aligned}$$

we eliminate u and get $x : X$ with $|p(x)|_1 = |a'|_1$ and, thanks to proposition 15 and another elimination, we get a path $q' : p(x) = a'$, thus getting $(x, q' \cdot r) : \mathsf{fib}_p(a)$. Then, we can define e' by

$$e'(v) :\equiv \mathsf{pr}_1(\mathsf{fib}\text{-}e'(\tilde{p}(v), (v, \mathsf{refl}_{\tilde{p}(v)}))).$$

We have that $e'(e(x)) = x$ holds definitionally. On the other hand, the proof that $e(e'(v)) = v$ needs more work. We can suppose $v \equiv (a, |x|_1, q)$ and that we have a path $q' : p(x) = a$ by truncation elimination and proposition 15. Then, we have that:

$$\begin{aligned} e(e'(v)) &\equiv e(\mathsf{pr}_1(\mathsf{fib}\text{-}e'(a, (v, \mathsf{refl}_a)))) \\ &= e(\mathsf{pr}_1(x, q' \cdot \mathsf{refl}_a)) \\ &\equiv e(x) \\ &\equiv (p(x), |x|, \mathsf{refl}_{|p(x)|_1}) \\ &= (a, |x|, q'). \end{aligned}$$

where the last equality is obtained by induction on q', abstracting over a. Thus concluding the construction of $r : X = C_{\pi_1(X)}$.

$$\begin{tikzcd} X \arrow[r, bend left, "e"] \arrow[r, equal, "r"] \arrow[rr, bend left=40, "|-|_1"] \arrow[dr, bend right, "p"'] & C_{\pi_1(X)} \arrow[l, bend left, "e'"] \arrow[r] \arrow[d, "\tilde{p}"] \arrow[dr, phantom, "\lrcorner", very near start] & \|X\|_1 \arrow[d, "\|p\|_1"] \\ & A \arrow[r, "|-|_1"] & \|A\|_1 \end{tikzcd} .$$

Next, we give a dependent path between $\tilde{p}$ and p:

$$\begin{aligned} \tilde{p} &\overset{dependent}{\rightsquigarrow} x \mapsto \mathsf{transport}(\mathsf{refl}, \tilde{p}(\mathsf{transport}(r, x))) \\ &= \tilde{p} \circ e \\ &\equiv p. \end{aligned}$$

The first equality is `funTypeTransp`, the second one is by the computation rule of ua.

By the computation rule of ua we also have that the equality of basepoints as

$$e'(\star_{C_{\pi_1(X)}}) \equiv e'(a, |x|, \mathsf{ap}_{|-|_1}(q^\star)) \equiv x.$$

Finally, it remains the equality of the "pointing equalities" (a dependent path between refl_a and $p^\star$) which is done, once again, by decomposing the two paths and treating them individually.

4.9. Galois Correspondence.

We can now conclude:

Theorem 2. For any path-connected[j] pointed type (A, a),

$$\mathsf{Subgroup}(\pi_1(A, a)) \simeq \mathsf{Covering}(A, a).$$

□

This proof has been fully verified in Cubical Agda and the source code is available on GitHub:

`https://github.com/hugo-s29/classifying-covering-types-agda/`.

Compiling this proof is very slow: expect about 4 to 5 hours of waiting. A Python script is provided to start multiple Agda instances to compile files in parallel when possible.

5. Conclusion.

In this proof of the Galois Correspondence, the use of categorical constructs (in this case, pullbacks) interpreted through the lens of Homotopy Type Theory was essential. It allowed us to shorten the proof and to abstract if from fairly technical topology-related topics. Could other

[j] By the way, in the proof, we didn't explicitly use that A was path-connected, but it is a hidden assumption as we are using $\|A\|_1$ as a delooping of $\pi_1(A, a)$.

theorems in Algebraic Topology be proven this way? Proofs like this one are more easily generalizable and extendable compared to the classic ones. These generalization give us more understanding into the (sometimes deep) links between topological constructs.

References

[CMO25] Camil Champin, Samuel Mimram, and Émile Oleon. "Delooping presented groups in homotopy type theory". Submitted. 2025. eprint: 2405.03264.

[Hat02] Allen Hatcher. *Algebraic topology*. Cambridge: Cambridge University Press, 2002, pp. xii+544. ISBN: 0-521-79160-X; 0-521-79540-0.

[HH18] Kuen-Bang Hou (Favonia) and Robert Harper. "Covering Spaces in Homotopy Type Theory". In: *22nd International Conference on Types for Proofs and Programs (TYPES 2016)*. Ed. by Silvia Ghilezan, Herman Geuvers, and Jelena Ivetic. Vol. 97. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018, 11:1–11:16. ISBN: 978-3-95977-065-1. DOI: 10.4230/LIPIcs.TYPES.2016.11. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TYPES.2016.11.

[MO24] Samuel Mimram and Émile Oleon. "Delooping cyclic groups with lens spaces in homotopy type theory". In: *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '24. Tallinn, Estonia: Association for Computing Machinery, 2024. ISBN: 9798400706608. DOI: 10.1145/3661814.3662077. eprint: 2405.10149. URL: https://doi.org/10.1145/3661814.3662077.

[MO25] Samuel Mimram and Émile Oleon. "Classifying covering types in homotopy type theory". Submitted. 2025.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013.

[WMP24] Jelle Wemmenhove, Cosmin Manea, and Jim Portegies. "Classification of Covering Spaces and Canonical Change of Basepoint". en. In: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. DOI: 10.4230/LIPICS.TYPES.2023.1. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TYPES.2023.1.

A. Context of this internship.

This internship was done in the *Cosynus* team of the LIX laboratory at École Polytechnique (*c.f.* figure 3) with, as of writing this, seven permanent members and twelve Ph.D. students. This team strongly focuses on the theory, and notably by using modelling data and processes as abstract models so correctness results, for example, can be proven. Some of the main themes in the team include dealing concurrent systems and numerical systems using abstract interpretation, geometric methods (including algebraic topology) and higher-dimensional categories. The LIX laboratory is a joint research unit with CRNS, INRIA and École Polytechnique.

The *Cosynus* team is part of the *proofs and algorithms* pole along with the *PARTOUT* (Proof Automation and RepresenTation: a fOundation of compUtation and deducTion), *AlCo* (Algorithms and Complexity) and *PhIQuS* (Physics, Information Theory, and Quantum Simulation) teams. Seminars are in common with the four teams and, as they are about different subjects, allow us to see the world of Mathematics and Computer Science from a different point of view.

Figure 3 | *Logos of the LIX Laboratory and École Polytechnique*

B. Some background in Category Theory.

Definition 18. A ***category*** $\mathbf{C}$ is defined as:

- a collection of ***objects*** written $\mathrm{Ob}_{\mathbf{C}}$;
- for every two objects A, B in $\mathbf{C}$, a collection of ***morphisms*** from A to B, written $\mathrm{Hom}_{\mathbf{C}}(A, B)$;
- for every three objects A, B, C in $\mathbf{C}$, a map

  $$- \circ - : \mathrm{Hom}_{\mathbf{C}}(A,B) \times \mathrm{Hom}_{\mathbf{C}}(B,C) \to \mathrm{Hom}_{\mathbf{C}}(A,C)$$

  called ***composition***;
- for every object A in $\mathbf{C}$, an ***identity*** morphism from A to A written id_A on A;

such that the following diagrams commute:

- for every $f : \mathrm{Hom}_{\mathbf{C}}(A,B)$, $g : \mathrm{Hom}_{\mathbf{C}}(B,C)$ and $h : \mathrm{Hom}_{\mathbf{C}}(C,D)$,

  $$\begin{tikzcd} A \arrow[r, "f"'] \arrow[rr, bend left, "g\circ f"] & B \arrow[r, "g"] \arrow[rr, bend right, "h\circ g"'] & C \arrow[r, "h"] & D \end{tikzcd},$$

  i.e. composition is associative;
- for every $f : \mathrm{Hom}_{\mathbf{C}}(A,B)$,

  $$\begin{tikzcd} A \arrow[r, "\mathrm{id}_A"'] \arrow[rr, bend left, "f"] & A \arrow[r, "f"] \arrow[rr, bend right, "f"'] & B \arrow[r, "\mathrm{id}_B"] & B \end{tikzcd},$$

  i.e. identities are neutral elements with respect to composition.

Example 9. The category **Top** is defined by:

- *objects*: topological spaces;
- *morphisms*: continuous functions (*a.k.a.* maps in this document);
- *composition*: function composition;
- *identity*: identity map.

Example 10. The category **Groups** is the category formed by:

- *objects*: groups;
- *morphisms*: group homomorphisms:
- *composition*: function composition;
- *identity*: identity homomorphism.

This is the category *of groups*.

Example 11. Fix some group G. The 1-object group category $\mathbf{B}\,G$ is defined this way:

- *objects*: one single object written $\bullet$;
- *morphisms*: for every element $g \in G$, a morphism $g : \bullet \to \bullet$;
- *composition*: $g \circ h := (gh) : \bullet \to \bullet$;
- *identity*: neutral element $1_G : \bullet \to \bullet$.

Here, the group *is* the category. It is in fact more than a group, it is also a ***groupoid*** (*c.f.* the following definition).

Definition 19. A ***groupoid*** is a category $\mathbf{G}$ such that every morphism is an isomorphism, that is, every $f : \mathrm{Hom}_{\mathbf{G}}(A,B)$ has an inverse $f^{-1} : \mathrm{Hom}_{\mathbf{G}}(B,A)$ such that

$$\begin{tikzcd} B \arrow[r, "f^{-1}"] \arrow[d, "\mathrm{id}_B"'] & A \arrow[d, "\mathrm{id}_A"] \arrow[dl, "f"] \\ B \arrow[r, "f^{-1}"'] & A \end{tikzcd}$$

commutes.

Proposition 20. Given a groupoid $\mathbf{G}$, for any object A in $\mathbf{G}$, the collection of automorphisms of A, written $\mathrm{Aut}_{\mathbf{G}}(A) := \mathrm{Hom}_{\mathbf{G}}(A,A)$, is a group. □

For the special case of the groupoid $\mathbf{B}\,G$, then $\mathrm{Aut}_{\mathbf{B}\,G}(\bullet) = G$.

Example 12. Given some topological space A, we define the ***fundamental groupoid*** $\mathbf{\Pi}_1(A)$ by:

- *objects*: elements of type A;
- *morphisms*: homotopy classes of paths in A;
- *composition*: $q \circ p := p \cdot q : x \rightsquigarrow z$ for $p : x \rightsquigarrow y$ and $q : y \rightsquigarrow z$;
- *identity*: for any $x : A$, the path $\mathrm{refl}_x : x \rightsquigarrow x$.

It is, in fact, a groupoid as every path is inversible.

We can recover the fundamental group using $\mathbf{\Pi}_1(X)$:

$$\pi_1(X,x) = \mathrm{Aut}_{\mathbf{\Pi}_1(X)}(x) = \mathrm{Hom}_{\mathbf{\Pi}_1(X)}(x,x).$$

Definition 20. A ***(covariant) functor*** $F : \mathbf{A} \to \mathbf{B}$ from $\mathbf{A}$ to $\mathbf{B}$ is given by:

- a map $F_{\mathrm{ob}} : \mathrm{Ob}_{\mathbf{A}} \to \mathrm{Ob}_{\mathbf{B}}$;
- for every objects A, B in $\mathbf{A}$, a map

  $$F_{\mathrm{hom}} : \mathrm{Hom}_{\mathbf{A}}(A,B) \to \mathrm{Hom}_{\mathbf{B}}(F_{\mathrm{ob}}(A), F_{\mathrm{ob}}(B));$$

such that the following diagrams commute:

$$\begin{tikzcd} F_{\mathrm{ob}}(A) \arrow[r, "F_{\mathrm{hom}}(f)"] \arrow[rr, bend right, "F_{\mathrm{hom}}(g\circ f)"'] & F_{\mathrm{ob}}(B) \arrow[r, "F_{\mathrm{hom}}(g)"] & F_{\mathrm{ob}}(C) \end{tikzcd}$$

$$\begin{tikzcd} F_{\mathrm{ob}}(A) \arrow[r, bend left, "\mathrm{id}_{F_{\mathrm{ob}}(A)}"] \arrow[r, bend right, "F_{\mathrm{hom}}(\mathrm{id}_A)"'] & F_{\mathrm{ob}}(A) \end{tikzcd}.$$

We will now write F for F_{ob} and F_{hom} as it is often unambiguous.

Another way of defining functors is that it maps a commutative diagram to a commutative diagram.

Example 13. For any group G, H, functors $F : \mathbf{B}G \to \mathbf{B}H$ are *exactly* group homomorphisms $G \to H$, as

- $F(\bullet_{\mathbf{B}G})$ can only be $\bullet_{\mathbf{B}H}$;
- $F(g \cdot h) = F(g \circ h) = F(g) \circ F(h) = F(g) \cdot F(h)$.

Example 14. We define the category **Groupoids** by:

- *objects*: groupoids;
- *morphisms*: functors;
- *composition*: functor composition (*i.e.* composing F_{ob} and F_{hom});
- *identity*: identity functor.

Example 15. Given a continuous map $f : A \to B$, then it induces a functor

$$\Pi_1(f) : \Pi_1(A) \to \Pi_1(B),$$

mapping $a \in A$ to $f(a)$ and homotopy classes of paths $[\gamma]$ to $[\gamma \circ f]$.

The fundamental group and fundamental groupoid are functors:

$$\Pi_1 : \mathbf{Top} \to \mathbf{Groupoids} \qquad \pi_1 : \mathbf{Top}_\star \to \mathbf{Groups},$$

where the category $\mathbf{Top}_\star$ is the category of *pointed topological spaces* (objects are pointed spaces, morphisms are pointed continuous functions).

Definition 21. In a category $\mathbf{C}$, the ***pullback*** of a diagram

$$\begin{tikzcd} & A \arrow[d, "f"] \\ B \arrow[r, "g"] & C \end{tikzcd}$$

is defined as an object $A \times_C B$ in $\mathbf{C}$ along with two morphisms a, b in $\mathbf{C}$ making

$$\begin{tikzcd} A \times_C B \arrow[r, "a"] \arrow[d, "b"] \arrow[dr, phantom, "\lrcorner", very near start] & A \arrow[d, "f"] \\ B \arrow[r, "g"] & C \end{tikzcd}$$

commute such that the following ***universal property*** holds:

for object W along with morphisms $\tilde{a}, \tilde{b}$ making

$$\begin{tikzcd} W \arrow[r, "\tilde{a}"] \arrow[d, "\tilde{b}"] & A \arrow[d, "f"] \\ B \arrow[r, "g"] & C \end{tikzcd}$$

commute there is a unique morphism $u : W \to A \times_C B$ such that

$$\begin{tikzcd} W \arrow[dr, dashed, "u"] \arrow[drr, bend left, "\tilde{a}"] \arrow[ddr, bend right, "\tilde{b}"'] & & \\ & A \times_C B \arrow[r, "a"] \arrow[d, "b"] \arrow[dr, phantom, "\lrcorner", very near start] & A \arrow[d, "f"] \\ & B \arrow[r, "g"] & C \end{tikzcd}$$

commutes.

Proposition 21. Any two pullbacks of the same diagram are isomorphic.

Proof. Let U and V be two pullbacks of the same diagram. By universal property of U and V we get two morphisms $x : U \to V$ and $y : V \to U$. By universal property of V (uniqueness of morphism), we have that $x \circ y : V \to V$ is the identity (as id_V also commutes the same diagram). By universal property of U (uniqueness of morphism), we have that $y \circ x : U \to U$ is the identity (as id_U also commutes the same diagram). Thus U and V are isomorphic. □

Example 16. In the category of HoTT-definable spaces, all pullbacks exists. Consider a diagram

$$\begin{tikzcd} & A \arrow[d, "f"] \\ B \arrow[r, "g"] & C \end{tikzcd},$$

then we consider the type

$$A \times_C B :\equiv \sum_{a:A} \sum_{b:B} f(a) = g(b).$$

This diagram thus commutes:

$$\begin{tikzcd} A \times_C B \arrow[r, "\mathrm{pr}_1"] \arrow[d, "\mathrm{pr}_2"] & A \arrow[d, "f"] \\ B \arrow[r, "g"] & C \end{tikzcd}.$$

Consider a type W along with morphisms $\tilde{a}, \tilde{b}$ making

$$\begin{tikzcd} W \arrow[r, "\tilde{a}"] \arrow[d, "\tilde{b}"] & A \arrow[d, "f"] \\ B \arrow[r, "g"] & C \end{tikzcd}$$

with an equality $h : \prod_{w:W} f(\tilde{a}(w)) = g(\tilde{b}(w))$, then we define, in a canonical way,

$$\begin{aligned} u : W &\longrightarrow A \times_C B \\ w &\longmapsto (\tilde{a}(w), \tilde{b}(w), h(w)). \end{aligned}$$

This is the unique[k] morphism such that the diagram

$$\begin{tikzcd} W \arrow[dr, dashed, "u"] \arrow[drr, bend left, "\tilde{a}"] \arrow[ddr, bend right, "\tilde{b}"'] & & \\ & A \times_C B \arrow[r, "a"] \arrow[d, "b"] & A \arrow[d, "f"] \\ & B \arrow[r, "g"] & C \end{tikzcd}$$

is commutative. We can now conclude that $A \times_C B$ is *the* (up to isomorphism) pullback of $A \xrightarrow{f} C \xleftarrow{g} B$.

[k]We could have used $h(w) \cdot \mathsf{refl}$ or any path operation that give refl but, by here "unique" means "up to path-homotopy". And we cannot define a non-trivial loop without more knowledge about A, B, C as those could be sets, or $\mathbb{S}^1$, or any other type.