1. Introduction

The goal of this lecture is to begin studying the "homotopy theory of types" using the basic axioms from the previous lectures. Other than the usual axioms for function types, Σ-types, Π-types, etc. our main tool will be the path induction axiom. We will see how path induction endows types with structures that resemble ∞-groupoid, or topological spaces. We have the following table of analogies:

<table>
<thead>
<tr>
<th>type theory</th>
<th>category theory</th>
<th>infinity groupoids</th>
<th>topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>groupoid</td>
<td>Kan complex</td>
<td>topological space</td>
</tr>
<tr>
<td>A : U</td>
<td>C</td>
<td>K</td>
<td>X</td>
</tr>
<tr>
<td>x : A</td>
<td>object c</td>
<td>x ∈ K₀</td>
<td>point x ∈ X</td>
</tr>
<tr>
<td>x =₀ y</td>
<td>Hom(x, y)</td>
<td>Map(x, y)</td>
<td>Path(x, y)</td>
</tr>
</tbody>
</table>

which we will try to justify. We will try to see how far one can go with this analogy using basic type theory with path induction alone. Near the end of the lecture, we will see where more input is needed, which will be given by the function extensionality and univalence axioms which will be discussed in future lectures.

Recall that path induction says the following: Given a type family

$$ C : \prod_{x,y \in A} (x = y) \to U $$

and a function

$$ c : \prod_{x \in A} C(x, x, \text{refl}_a) $$

there is a function

$$ f : \prod_{x, y \in A} \prod_{p : x = y} C(x, y, p) $$

such that $f(x, x, \text{refl}_x) \equiv c(x)$.

2. Types are higher groupoids

We start with the easiest application of the induction principle by proving that inverses of paths exist.

**Lemma 2.0.1.** For every type A and every x, y : A there is a function

$$(x = y) \to (y = x)$$

denoted $p \mapsto p^{-1}$, such that $\text{refl}_x^{-1} \equiv \text{refl}_x$ for each $x : A$. We call $p^{-1}$ the inverse of $p$.

**Proof.** Recall that in our language this means that the type

$$ \prod_{(A : U)} \prod_{x : y \in A} (x = y) \to (y = x). $$

Lemma 2.0.3. Suppose lemma. Composition and inverse satisfy the expected compatibility conditions, as shown by the following:

Let \( D : \prod_{(x,y:A)}(x = y) \rightarrow \mathcal{U} \) be the type family defined by \( D(x, y, p) \equiv (y = x) \). By induction, it is enough to consider the family \( D(x, x, \text{refl}_x) \). Then we have an element

\[
d \equiv \lambda x. \text{refl}_x : \prod_{x:A} D(x, x, \text{refl}_x).
\]

This gives an element \((-1) : \prod_{(x,y:A)} \prod_{p:x=y} D(x, y, p) \) such that \( \text{refl}_x^{-1} \equiv \text{refl}_x \).

We can use a similar technique to construct compositions, or in the topological language, concatenation of paths.

Lemma 2.0.2. For every type \( A \) and \( x, y : A \) there is a function

\[
(x = y) \rightarrow (y = z) \rightarrow (x = z)
\]

which we write as \( p \mapsto q \rightarrow p \circ q \), such that \( \text{refl}_x \circ \text{refl}_x \equiv \text{refl}_x \) for any \( x : A \). We call \( p \circ q \) the composition of \( p \) and \( q \).

Proof. Let \( D : \prod_{(x,y:A)}(x = y) \rightarrow \mathcal{U} \) be the type family

\[
D(x, y, p) \equiv \prod_{z:A} \prod_{q:y=z} (x = z).
\]

To get an inhabitant of this type via induction we need a function of type

\[
\prod_{x:A} D(x, x, \text{refl}_x) = \prod_{(x,z:A)} \prod_{q:x=z} (x = z)
\]

Let \( E : \prod_{(x,z:A)}(x = z) \rightarrow \mathcal{U} \) be the type family \( E(x, z, q) \equiv (x = z) \). As \( E(x, x, \text{refl}_x) \equiv (x = x) \) we have the function

\[
e(x) \equiv \text{refl}_x : E(x, x, \text{refl}_x)
\]

Path induction now gives us a \( d : \prod_{(x,y,z:A)}(y = z) \rightarrow (x = y) \rightarrow (x = z) \) with \( d(x, x, \text{refl}_x) \equiv \text{refl}_x \). Thus, applying path induction to \( D \), we get a function of type

\[
\prod_{(x,y,z:A)} (y = z) \rightarrow (x = y) \rightarrow (x = z).
\]

Composition and inverse satisfy the expected compatibility conditions, as shown by the following lemma.

Lemma 2.0.3. Suppose \( A : \mathcal{U} \) and \( x, y, z, w : A \) and that \( p : x = y \), \( q : y = z \), and \( r : z = w \) are 1-paths. We have the following:

1. \( p = p \circ \text{refl}_y \) and \( p \equiv \text{refl}_x \circ p \).
2. \( p^{-1} \circ p = \text{refl}_x \) and \( p \circ p^{-1} = \text{refl}_y \).
3. \( (p^{-1})^{-1} = p \).
4. \((p \circ q) \circ r = p \circ (q \circ r)\).

Proof. We will prove these claims by induction.

(1) Let \( D \) be the type family \( D(x, y, p) \equiv (p = p \circ \text{refl}_y) \). As \( D(x, x, \text{refl}_x) = (\text{refl}_x \circ \text{refl}_x = \text{refl}_x) \) and we have \( \text{refl}_x \circ \text{refl}_x \equiv \text{refl}_x \), there is a function

\[
d \equiv \lambda x. \text{refl}_x \circ \text{refl}_x : \prod_{x:A} D(x, x, \text{refl}_x).
\]

Induction on \( D \) gives a type \( \prod_{(x,y:A)} \prod_{p:x=y} (p = p \circ \text{refl}_y) \). The other direction is proved similarly.
(2) Let \( D(x, y, p) = (p^{-1} \circ p = \text{refl}_x) \) be a type family. By induction, it is enough to inhibit \( \prod_{x \in A} (\text{refl}_x^{-1} \circ \text{refl}_x = \text{refl}_x) \). As \( \text{refl}_x^{-1} \circ \text{refl}_x \) is judgmentally equal to \( \text{refl}_x \), we can get such a type by the function \( \Lambda x. \text{refl}_x^{-1} \).

(3) By induction, we can assume \( p \) is \( \text{refl}_x \), and in this case we have \( (\text{refl}_x^{-1})^{-1} = \text{refl}_x \).

(4) Consider the type family
\[
D_1(x, y, p) \equiv \prod_{(z, w : A)} \prod_{(q : y = z)} \prod_{(r : z = w)} ((p \circ q) \circ r = p \circ (q \circ r)).
\]
By induction, we can consider
\[
D_1(x, x, \text{refl}_x) \equiv \prod_{(w : A)} \prod_{(r : w = w)} ((\text{refl}_x \circ q) \circ r = \text{refl}_x \circ (q \circ r)).
\]
To construct an element of this type, consider the type family
\[
D_2(x, z, q) \equiv \prod_{(w : A)} ((\text{refl}_x \circ q) \circ r = \text{refl}_x \circ (q \circ r))
\]
and by induction, we can consider
\[
D_2(x, x, \text{refl}_x) \equiv \prod_{(w : A)} ((\text{refl}_x \circ \text{refl}_x) \circ r = \text{refl}_x \circ (\text{refl}_x \circ r)).
\]
Finally, we consider the family
\[
D_3(x, w, r) \equiv \prod_{(w : A)} ((\text{refl}_x \circ \text{refl}_x) \circ r = \text{refl}_x \circ (\text{refl}_x \circ r))
\]
then
\[
D_3(x, x, \text{refl}_x) \equiv \prod_{(x : A)} ((\text{refl}_x \circ \text{refl}_x) \circ \text{refl}_x = \text{refl}_x \circ (\text{refl}_x \circ \text{refl}_x))
\]
Since \( \text{refl}_x \circ \text{refl}_x \equiv \text{refl}_x \) we get a judgmental equality \( D_3(x, x, \text{refl}_x) = \prod_{(x : A)} (\text{refl}_x = \text{refl}_x) \) which is inhibited by \( \Lambda x. \text{refl}_x^{-1} : D_3(x, x, \text{refl}_x) \).

**Remark 2.0.1.** The proof above does more than

We have seen how path induction can be used to define composition of paths (or morphisms), inverses, and that these satisfy the usual relations. If these were set valued functions, there would be no other structure to explore as all equalities are by definition ”judgmental”. However, this is not the case for types, and similarly, higher groupoids. In a higher groupoid, for any triple of composable 1-morphisms, there is a whole space of ways to compose them, as we can see by studying Kan simplicial sets for example. This richer structure also exists in types since we now have an entire type of equalities \( (p \circ q) \circ r = p \circ (q \circ r) \). Moreover, it is inherent in the language of type theory that such higher structure exists. If we wish to argue that types do indeed behave like higher groupoids, we would like to see the coherent conditions for higher morphisms appear. We will see an example for such behavior in theorem 2.0.1. Before we can do that, we need a bit of terminology.

A **pointed type** \((A, a)\) is a type \( A : U \) together with a point \( a : U \) called its basepoint. We write \( U_* = \sum_{A : U} A \) for the type of pointed types in the universe \( U \).

**Definition 2.0.1.** Given a pointed type \((A, a)\) we define the loop space to be the following pointed type:
\[
\Omega(A, a) = ((a = A), \text{refl}_A)
\]
The \( n \)-fold iterated loop space \( \Omega^n(A, a) \) is defined recursively as
\[
\Omega^{n+1}(A, a) = \Omega(\Omega^n (A, a))
\]
Theorem 2.0.1 (Eckmann-Hilton). The composition operation on the second loop space
\[ \Omega^2(A) \times \Omega^2(A) \rightarrow \Omega^2(A) \]
is commutative. i.e. \( \alpha \circ \beta = \beta \circ \alpha \) for any \( \alpha, \beta \in \Omega^2(A) \).

**Proof.** First, note that composition of 1-loops \( \Omega(A) \times \Omega(A) \rightarrow \Omega(A) \) induces an operation
\[ \ast : \Omega^2(A) \times \Omega^2(A) \rightarrow \Omega^2(A) \]
as follows. For \( a, b, c \in A \) and paths
\[
\begin{align*}
p : a &= b, r : b = c \\
q : a &= b, r : b = c \\
\alpha : p &= q, \beta : r = s
\end{align*}
\]
depicted as follows.

![Diagram](image)

We can compose the 1-paths to get \( p \circ r, q \circ s : a = c \). We then define \( \alpha \ast \beta : p \circ r = q \circ s \) as follows. First, define \( \alpha \circ_r r : p \circ r = q \circ r \) by path induction on \( r \) given by
\[
\alpha \circ_r \text{refl}_b \equiv \text{ru}^{-1}_p \circ \alpha \circ \text{ru}_q
\]
where \( \text{ru}_p : p = p \circ \text{refl}_b \) is right unit law from lemma 2.0.3. Similarly, we define \( q \circ_l \beta : q \circ r = q \circ s \) by induction on \( q \) so that
\[
\text{refl}_b \circ_l \beta \equiv \text{lu}^{-1}_r \circ \beta \circ \text{lu}_a
\]
where \( \text{lu}_r \) denotes the left unit law. The operations \( \circ_r \) and \( \circ_l \) are called whiskering. We now define
\[
\alpha \ast \beta \equiv (\alpha \circ_r r) \circ (q \circ_l \beta)
\]
Now suppose that \( a \equiv b \equiv c \) and that \( p \equiv q \equiv r \equiv s \equiv \text{refl}_a \) so that we have \( \alpha : \text{refl}_a = \text{refl}_a \) and \( \beta : \text{refl}_a = \text{refl}_a \). Then we have
\[
\begin{align*}
\alpha \ast \beta &\equiv (\alpha \circ_r \text{refl}_a) \circ (\text{refl}_a \circ_l \beta) \\
&= \text{ru}_{\text{refl}_a}^{-1} \circ \alpha \circ \text{ru}_{\text{refl}_a} \circ \text{lu}_{\text{refl}_a}^{-1} \circ \beta \circ \text{lu}_{\text{refl}_a} \\
&\equiv \text{refl}_{\text{refl}_a}^{-1} \circ \alpha \circ \text{refl}_{\text{refl}_a} \circ \text{refl}_{\text{refl}_a}^{-1} \circ \beta \circ \text{refl}_{\text{refl}_a} \\
&\equiv \alpha \circ \beta.
\end{align*}
\]
Similarly, we can define whiskering in the other direction by
\[
\alpha \ast' \beta \equiv (p \circ_l \beta) \circ (\alpha \circ_r s)
\]
and we similarly get that \( \alpha \ast' \beta = \beta \circ \alpha \). However, we now argue that in fact \( \alpha \ast \beta = \alpha \ast' \beta \) as we can see by induction on \( \alpha, \beta \) as follows. We consider the type family \( D : \Pi_{(p,q,a=\text{refl}_a)} (p = q) \rightarrow \mathcal{U} \) given by
\[
D(p,q,\alpha) \equiv \prod_{r,s,b=c} (\alpha \ast \beta = \alpha \ast' \beta)
\]
By path induction, it is enough to construct an element of
\[
D(p,p,\text{refl}_p) \equiv \prod_{r,s,b=c} (\text{refl}_p \ast \beta = \text{refl}_p \ast' \beta)
\]
To do that, consider the type family \( C : \Pi_{r,s,b,c} (r = s) \rightarrow \mathcal{U} \) given by
\[
C(r,s,\beta) \equiv (\text{refl}_p \ast \beta = \text{refl}_p \ast' \beta).
\]
Then,
\[
C(r,r,\text{refl}_r) = \prod_{r,s,b=c} (\text{refl}_p \ast \text{refl}_r = \text{refl}_p \ast' \text{refl}_r).
\]
For this, we define a dependent function \( E : \prod_{(a, b : A)} (a = b) \to \mathcal{U} \) by
\[
E(a, b, p) \equiv \text{refl}_p \star \text{refl}_r = \text{refl}_p \star' \text{refl}_r
\]
Then,
\[
E(a, a, \text{refl}_a) \equiv \text{refl}_{\text{refl}_a} \star \text{refl}_r = \text{refl}_{\text{refl}_a} \star' \text{refl}_r
\]
We can now do induction on \( r \) to get that it is enough to present a type of the form
\[
\equiv \text{refl}_{\text{refl}_a} \star \text{refl}_{\text{refl}_a} = \text{refl}_{\text{refl}_a} \star' \text{refl}_{\text{refl}_a}
\]
which reduces to reflexivity as \( \text{refl}_{\text{refl}_a} \star \text{refl}_{\text{refl}_a} \equiv \text{refl}_{\text{refl}_a} \) and \( \text{refl}_{\text{refl}_a} \star' \text{refl}_{\text{refl}_a} \equiv \text{refl}_{\text{refl}_a} \).

3. Functions are functors

In the last section we have seen that types share some features we would like a higher groupoid to have, such as composition of morphisms, inversion, and even the commutativity of higher pointed loop spaces. In this short section we will see that function types behave functorially.

**Lemma 3.0.1.** Let \( f : A \to B \) be a function. Then for any \( x, y : A \) there is a function
\[
\text{ap}_f : (x = A y) \to (f(x) = B f(y))
\]
such that \( f(\text{refl}_x) \equiv \text{refl}_{f(x)} \).

**Proof.** Let \( D : \prod_{(x, y : A)} (x = y) \to \mathcal{U} \) be defined by
\[
D(x, y, p) \equiv (f(x) = f(y)).
\]
Then \( D(x, x, \text{refl}_a) = (f(x) = f(x)) \) which we can take to be \( \text{refl}_{f(x)} \).

We will often write \( f(p) \) for \( \text{ap}_f(p) \). Similar to the proves in the previous section, one can show the following.

**Lemma 3.0.2.** For functions \( f : A \to B, g : B \to C \) and morphisms (or paths) \( p : x =_A y \) and \( q : y =_A z \), we have

1. \( f(p \circ q) = f(p) \circ f(q) \).
2. \( f(p^{-1}) = f(p)^{-1} \).
3. \( g(f(p)) = (g \circ f)(p) \).
4. \( \text{id}_A(p) = p \).

4. Type families are fibrations

In the theory of infinity groupoids (or \((\infty,0)\)-categories), the straightening/unstraightening constructions allows us to identify \((\text{Kan})\)-fibrations \( f : E \to K \) with functors \( f : K \to \infty - \text{Grpd} \) by sending a point \( x \in X \) to the fiber \( f^{-1}(x) \) and a morphism \( x \to y \) to some lifting, which is unique up to homotopy, and obtained using the Kan lifting property.

In topology, recall that a map \( \pi : E \to B \) of topological spaces is called a fibration if it satisfies the homotopy lifting property with respect to any space. That is, given a homotopy \( f : X \times I \to B \) and a map \( f_0 : X \to E \) lifting \( f_0 \), one can find a lifting
\[
\begin{array}{ccc}
X & \xrightarrow{f_0} & E \\
\downarrow & & \downarrow \pi \\
X \times I & \xrightarrow{f} & B
\end{array}
\]
This allows us to lift paths from \( B \) to \( E \). Unlike the case of covering spaces though (which correspond to usual categories) the lifting is not necessarily unique.

In type theory, we have a clear analogue of a functor to spaces. It will simply be a type family \( P : A \rightarrow \mathcal{U} \). The fiber over \( x : A \) would then be the type \( P(x) \). We can now ask whether there is an analogous fibration corresponding to \( P \). To answer that question, the first thing we would do is try to lift paths (or morphisms) from \( A \).

**Lemma 4.0.1** (Transport). Suppose that \( P \) is a type family over \( A \) and that \( p : x = A y \). Then there is a function \( f_* : P(x) \rightarrow P(y) \).

*Proof.* Let \( D : \prod_{(x,y:A)}(x = y) \rightarrow \mathcal{U} \) be the type family defined by

\[
D(x,y,p) \equiv P(x) \rightarrow P(y).
\]

Then \( D(x,x,\text{refl}_x) \equiv P(x) \rightarrow P(x) \) and we can get an element via

\[
\lambda x. \text{id}_{P(x)} : \prod_{x:A} D(x,x,\text{refl}_x)
\]

Path induction then gives us a desired inhabitant of \( \prod_{x,y:A} \prod_{p : x = A y} D(x,y,p) \), which is proof of the proposition. \( \square \)

**Lemma 4.0.2** (Path lifting property). Let \( P : A \rightarrow \mathcal{U} \) be a type family over \( A \) and let \( u : P(x) \) for some \( x : A \). Then for any \( p : x = y \) we have

\[
\text{lift}(u,p) : (x,u) \equiv (y,p_*(u))
\]

in \( \sum_{x:A} P(x) \) such that \( \text{pr}_1(\text{lift}(u,p)) = p \).

*Proof.* Consider the type family \( D : \text{prod}_{(x,y:A)}(x = y) \rightarrow \mathcal{U} \) given by

\[
D(x,y,p) \equiv \prod_{u : P(x)} (x,u) = (y,p_*(u)).
\]

Then, \( D(x,x,\text{refl}_x) \equiv \prod_{u : P(x)} (x,u) = (x,\text{refl}_x)_*(u) \). As \( (\text{refl}_x)_* \equiv \text{id}_{P(x)} \) we can take \( \lambda x. \text{refl}(x,u) : \prod_{x:A} D(x,x,\text{refl}_x) \), and conclude by path induction. \( \square \)

Thus, we can think of a type family \( P : A \rightarrow \mathcal{U} \) as corresponding to the fibration \( \text{pr}_1 : \Sigma_{(x:A)} P(x) \rightarrow A \) with total space \( \Sigma_{(x:A)} P(x) \) and fiber \( P(x) \). In this picture the space of sections is modeled by the product \( \prod_{x:A} P(x) \). In that case, given \( f : \prod_{x:A} P(x) \) we define a function \( f' : A \rightarrow \Sigma_{x:A} P(x) \) by setting \( f'(x) \equiv (x,f(x)) \).

**Lemma 4.0.3** (Dependent map). Suppose \( f : \prod_{(x:A)} P(x) \). Then we have a map

\[
\text{adp}_f : \prod_{(p : x=y)} (p_*(f(x)) =_{P(y)} f(y))
\]

*Proof.* Consider the type family \( D(x,y,p) \equiv (p_*(f(x)) = f(y)) \). Then \( D(x,x,\text{refl}_x) \) is judgmentally \( (\text{refl}_x)_*(f(x)) \equiv f(x) \). Since \( (\text{refl}_x)_*(f(x)) \equiv f(x) \) we can take

\[
\lambda x. \text{refl}_x(f(x)) : \prod_{(x:A)} D(x,x,\text{refl}_x).
\]

The desired map then exists by path induction. \( \square \)

**Lemma 4.0.4.** If \( P : A \rightarrow \mathcal{U} \) is given by \( P(x) \equiv B \) for a fixed \( B : \mathcal{U} \) (so that \( B(x) \equiv B(y) \) for all \( x,y : A \)), then for any \( x,y : A, p : x = y \) and \( b : B \) we have a morphism

\[
p_*(b) : p_*(b) = b.
\]
Thus, for any \( x, y \) which is judgmentally \( b \)

\[
\prod_{x:A} (\text{refl}_x)_*(b) = b,
\]

which is judgmentally \( b = b \) so we can take \( d(x) \equiv \text{refl}_x \). □

Thus, for any \( x, y : A \) and \( f : A \to B \), we obtain functions

\[
\begin{align*}
(f(x) = f(y)) & \to (p_*(f(x)) = f(y) \\
(p_*(f(x)) = f(y) & \to (f(x) = f(y)).
\end{align*}
\]

We would like to say that these functions are inverse equivalences (in some sense). This will be defined later today.

**Lemma 4.0.5.** Given \( P : A \to \mathcal{U} \) with \( p : x = y, q : y = z \) and \( u : P(x) \), we have

\[
q_*(p_*u) = (p \circ q)_*(u).
\]

**Lemma 4.0.6.** For a function \( f : A \to B \) and a type family \( P : B \to \mathcal{U} \), for any map \( p : x =_A y \) and \( u : P(f(x)) \) we have

\[
p_*(u) = f(p)_*(u),
\]

where \( p_* : P(f(x)) \to P(f(y)) \) is the transport map for the type family \( B \circ f : A \to \mathcal{U} \).

**Lemma 4.0.7.** For \( P, Q : A \to \mathcal{U} \) and a family of functions \( \prod_{x:A} P(x) \to Q(x) \), and any \( p : x =_A y, u : P(x) \) we have

\[
p_*(f_*(u)) = f_*(p_*(u)).
\]

5. Homotopies and equivalences

In the theory of infinity groupoids, we think of functions between spaces as functors. Under this
dictionary, a homotopy corresponds to a natural transformation. There is an analogous picture in type
theory.

**Definition 5.0.1.** Let \( f, g : \prod_{x:A} P(x) \) be two sections of a type family \( P : A \to \mathcal{U} \). A homotopy from \( f \) to \( g \) is a dependent function of type

\[
(f \sim g) \equiv \prod_{(x:A)} (f(x) = g(x)).
\]

**Remark 5.0.1.** Note that a homotopy is not the same as \( f = g \). In next week’s lecture, we will introduce
an axiom making homotopies and identifications “equivalent” called ”function extensionality”.

**Lemma 5.0.1 (Homotopy is an equivalence relation).** We have types

\[
\begin{align*}
\prod_{f : \prod_{x:A} P(x)} (f \sim f) \\
\prod_{f,g : \prod_{x:A} P(x)} (f \sim g) \to (g \sim f) \\
\prod_{f,g,h : \prod_{x:A} P(x)} (f \sim g) \to (g \sim h) \to (f \sim h).
\end{align*}
\]

We would like to say that homotopies behave like natural transformations.

**Lemma 5.0.2 (Naturality).** Suppose \( H : f \simeq g \) is a homotopy between \( f, g : A \to B \) and let \( p : x =_A y \). Then we have

\[
H(x) \circ g(p) = f(p) \circ H(y).
\]
Proof. Consider the family \( D : \prod_{x,y:A} (x = y) \to \mathcal{U} \) given by
\[
D(x, y, p) \equiv (H(x) \circ g(p) = f(p) \circ H(y))
\]
By induction, we may consider \( H(x) \circ g(\text{refl}_x) = f(\text{refl}_x) \circ H(x) \). As \( f(\text{refl}_x) \equiv \text{refl}_{f(x)} \) and likewise for \( g \), we need to present an element of
\[
H(x) \circ \text{refl}_{g(x)} = \text{refl}_{f(x)} \circ H(x).
\]
As both sides are judgmentally equal to \( H(x) \), we can take \( \text{refl}_{H(x)} \).

As is usually done in category theory, we can draw the naturality relation as a diagram

\[
\begin{array}{ccc}
  f(x) & \overset{f(y)}{\longrightarrow} & f(y) \\
  \downarrow{H(z)} & & \downarrow{H(y)} \\
  g(x) & \overset{g(y)}{\longrightarrow} & g(y).
\end{array}
\]

**Corollary 5.0.1.** Let \( H : f \simeq \text{id}_A \) be a homotopy with \( f : A \to A \). Then for any \( x : A \) we have
\[
H(f(x)) = f(H(x)).
\]

**Proof.** The naturality of \( H \) with respect to \( H(x) : f(x) = x \) implies the following

\[
\begin{array}{ccc}
  f(f(x)) & \overset{f(H(x))}{\longrightarrow} & f(x) \\
  \downarrow{H(f(x))} & & \downarrow{H(x)} \\
  f(x) & \overset{H(x)}{\longrightarrow} & x
\end{array}
\]

commutes. That is, there is a type \( f(H(x)) \circ H(x) = H(f(x)) \circ H(x) \). We can now use whiskering and apply \( H(x)^{-1} \) to obtain
\[
f(H(x)) = f(H(x)) \circ H(x) \circ H(x)^{-1} = H(f(x)) \circ H(x) \circ H(x) = H(f(x)).
\]
where we applied associativity along the way. \( \square \)

**Definition 5.0.2.** For a function \( f : A \to B \), a quasi-inverse of \( f \) is a triple \((g, \alpha, \beta)\) consisting of \( g : B \to A \) and homotopies \( \alpha : f \circ g \sim \text{id}_B \) and \( \beta : g \circ f \sim \text{id}_A \). We denote the type of quasi-inverses of \( f \) by \( \text{qinv}(f) \).

However, in order to parametrize the proposition "\( f : A \to B \) is a homotopy equivalence", it turns out that \( \text{qinv}(f) \) is not well behaved. For example, there could be multiple unequal inhabitants of \( \text{qinv}(f) \). The reason for this is that we do not yet have enough constraints in our theory to force different inverses to be equal. Instead, we willmake the following definition

**Definition 5.0.3.**
\[
\text{isequiv}(f) \equiv \left( \sum_{g : B \to A} (f \circ g) \sim \text{id}_B \right) \times \left( \sum_{h : A \to B} (h \circ f) \sim \text{id}_A \right)
\]

In contrast with \( \text{qinv}(f) \), any two inhabitants of \( \text{isequiv}(f) \) are equal. This will be discussed in later lectures.

One can construct inhabitants of \( \text{qinv}(f) \to \text{isequiv}(f) \) and \( \text{isequiv}(f) \to \text{qinv}(f) \) so the two are logically equivalent. Indeed, we can define a function \( \text{qinv}(f) \to \text{isequiv}(f) \) by sending \((g, \alpha, \beta)\) to \((g, \alpha, g, \beta)\). On the other direction, given \((g, \alpha, h, \beta) \in \text{isequiv}(f)\) we let \( \gamma \) be the composite homotopy
\[
g \overset{\beta}{\sim} h \circ f \circ g \overset{\alpha}{\sim} h
\]
meaning that \( \gamma(x) \equiv \beta(g(x))^{-1} \circ h(\alpha(x)) \). Now define \( \beta' : g \circ f \sim \text{id}_A \) by \( \beta'(x) = \gamma(f(x)) \circ \beta(x) \). Then \( (g, \alpha, \beta') : \text{qinv}(f) \).

**Definition 5.0.4.** For \( A, B : \mathcal{U} \) we define the type of equivalences from \( A \) to \( B \) by

\[
A \simeq B \equiv \sum_{f : A \to B} \text{isoequiv}(f).
\]

**Proposition 5.0.1.** Type equivalence is an equivalence relation on \( \mathcal{U} \). That is,

1. For any \( A : \mathcal{U} \), \( \text{id}_A \) is an equivalence, so \( A \simeq A \).
2. For any \( f : A \simeq B \), we have an equivalence \( f^{-1} : B \simeq A \).
3. For any \( f : A \simeq B \) and \( g : B \simeq C \), we have an equivalence \( g \circ f : A \simeq B \).

**Proof.**

(i) The identity function is clearly a quasi-inverse of itself, so \( A \simeq A \) by \( \text{id}_A \).

(ii). If \( f : A \to B \) is an equivalence, it has a quasi-inverse, say \( f^{-1} : B \to A \). Then \( f \) is also a quasi-inverse of \( f^{-1} \), so \( f^{-1} \) is an equivalence \( B \simeq A \).

(iii). Given \( f : A \simeq B \) and \( g : B \simeq C \), with quasi-inverses \( f^{-1} : B \to A \) and \( g^{-1} : C \to B \), for any \( a : A \) we have \( f^{-1}g^{-1}gf(a) = f^{-1}f(a) = a \) and for any \( c : C \) we have \( gff^{-1}g^{-1}(c) = gg^{-1}c = c \). Hence, \( f^{-1} \circ g^{-1} \) is a quasi-inverse to \( g \circ f \). \( \square \)