Synthetic Homotopy Theory

Lecture notes

Proof assistants and applications
31 August – 4 September 2026, Strasbourg, France

by
Axel Ljungström
University of Strathclyde

Lecture 1

NOTE: These notes are not very polished and may contain errors and (and some unfinished todos). Please take them with a grain of salt. They should contain all info you need for the multiple choice quiz, however.

What is homotopy type theory?

In this course, we'll use type theory to reason about *homotopy theory*. The type theory we'll use is MLTT (as in Loïc's course) equipped with two additional axioms/constructions:

- Univalence: 'equivalent types are equal' (more details later)
  - The torus is equal to the product of two circles $S^1 \times S^1$
  - Any two bijective sets are equal
  - Etc...
- Higher inductive types: a generalisation of inductive types that allow us to capture certain spaces

We'll introduce these concepts more as we go. For now, let's stick to the basic type theory we all know and love. We'll use the following notational conventions:

- We will write $A : \mathsf{Type}$ for 'A is a type'. Here, Type denotes the *universe*, i.e. the type of all types*
- Dependent types are written $B : A \to \mathsf{Type}$.
- $\Pi_{x:A} B(x)$ is often written $(x : A) \to B(x)$
- $\Sigma_{x:A} B(x)$ is often written $(x : A) \times B(x)$

A crash course in traditional homotopy theory

As this course is about using type theory to reason about homotopy theory, we should probably quickly recall what kinds of mathematics the latter field is actually concerned with.

In traditional homotopy theory/topology, a path between two points $x, y \in A$ is formally defined as a continuous path $p : [0,1] \to A$ s.t. $p(0) = x$ and $p(1) = y$. Let's write $\mathsf{Path}_A(x,y)$ for the set of such paths. We can compose and invert paths – given $p \in \mathsf{Path}_A(x,y)$ and $\mathsf{Path}_A(y,z)$, we can define

$$(p \cdot q)(x) :\equiv \begin{cases} p(2x) & \text{if } x < 0.5 \\ q(2x-1) & \text{if } x \geq 0.5 \end{cases}$$
$$p^{-1}(x) :\equiv p(1-x)$$

Given two paths $p, q \in \mathsf{Path}_A(x,y)$, we can also capture the notion of a continuous transformation between these by saying that there is another continuous function $r : [0,1]^2 \to A$ .s.t. $r(0,x) = p(x)$ and $r(1,x) = q(x)$ for all $x \in [0,1]$. Such a transformation is called a *homotopy* and is denoted $p \sim q$. This can be generalised: we can have homotopies between homotopies (and so on) by replacing $[0,1]^2$ by $[0,1]^n$ for any $n > 2$. Some homotopies are especially central to this course:

Lemma. *Let $p \in \mathsf{Path}_A(x,y)$, $q \in \mathsf{Path}_A(y,z)$, and $r \in \mathsf{Path}_A(z,w)$. We have*

- $p \cdot \mathsf{const}_y \sim p$
- $\mathsf{const}_x \cdot p \sim p$
- $p \cdot p^{-1} \sim \mathsf{const}_x$
- $p^{-1} \cdot p \sim \mathsf{const}_x$
- $(p \cdot q) \cdot r \sim p \cdot (q \cdot r)$

The notion of homotopy easily generalises to arbitrary continuous functions rather than just paths: we say that two functions $f, g : X \to Y$ are homotopic if there is another continuous function $h : I \times X \to Y$ s.t. $h(0,x) = f(x)$ and $h(1,x) = g(x)$ for all $x \in X$. We write $f \sim g$ to denote the existence of a homotopy between f and g.

We say that two spaces are *homotopy equivalent* if we have maps $f : X \to Y$ and $g : Y \to X$ s.t. $f \circ g \sim \mathsf{id}$ and $g \circ f \sim \mathsf{id}$. In the field of homotopy theory, we study paths and spaces up to homotopy (equivalence).

One thing that homotopy theorists like to do is to restrict to a certain class of paths, namely those that are *loops*, i.e. start and end at the same point.

Draw pic

Given a space A and $x \in A$, we can define

$$\Omega(A,x) \equiv \{\text{loops in } A \text{ starting and ending at } x\}$$

Path composition and inversion define continuous operations on this space. Furthermore, it has a 'neutral element', namely the constant path at x. Because path composition satisfies all the group laws up to homotopy (previous lemma), this makes $\Omega(A,x)$ into 'group' – up to homotopy.

Remark. We can turn this into an actual group by quotienting out by homotopy:

$$\pi_1(A, x) := \Omega(A, x)/\sim$$

This is called the *fundamental group of A (at x)*. We think of $\pi_1(A, x)$ as the *set* of equivalences classes of loops in $\Omega(A, x)$. This is an important difference from $\Omega(A, x)$ which itself is not viewed as a *set* but as a *topological space*.

Remark. We can iterate Ω to get the nth loop space of A, i.e. $\Omega^n(A, x)$. This allows us to generalise π_1. Let

$$\pi_n(A, x) := \Omega^n(A, x)/\sim$$

This is called the nth homotopy group and is one of the key objects of study in homotopy theory.

It's very hard to compute $\Omega(X)$ of a space in full detail. It's usually a bit easier to compute π_1 (but in general extremely hard). However, in this course we'll focus on a case where these two definitions happen to coincide (up to homotopy equivalence). The goal of this course is to show that $\Omega(S^1)$, i.e. of the loop space of the circle, is equivalent to the integers.

What does this have to do with type theory?

The set interpretation of type theory Types are *not* sets. However, everything we do in type theory can be *interpreted* as being about sets. For example:

- The **type** $\mathbb{N}$ can be interpreted as the set $\{0, 1, 2, \dots\}$
- When we write $2 : \mathbb{N}$, we can interpret this as the statement $2 \in \{0, 1, 2, \dots\}$
- When we write $a = b$ in type theory, we can interpret this as the set

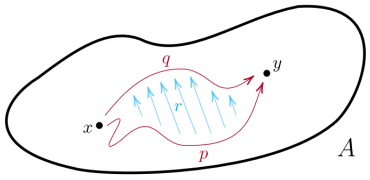
$$X :\equiv \begin{cases} \{*\} & \text{if } a = b \\ \emptyset & \text{if } a \neq b \end{cases}$$

The idea is that, since all of classical mathematics is about sets, and types can be interpreted as sets, we can use type theory to reason about all of classical mathematics.

Another interpretation So, types can be interpreted as sets – great, but why sets and not something else? Mathematicians like Awodey, Hoffmann, Streicher, Voevodsky, and Warren asked this question and came up with an alternative answer: types can be interpreted as spaces[1].

[1] Voevodsky is the person who is often directly attributed this connection, but the other authors have also made crucial contributions that inspired this interpretation

- when we write that 'A is a type' (or '$A : \mathsf{Type}$'), we interpret this as 'A is a topological space (up to homotopy equivalence),
- when we write that '$x : A$', we interpret this as 'x is a point in A',
- when we write that '$p : x =_A y$', we interpret this as 'p is a (continuous) path from x to y in A',
- when we write that $r : p =_{x=y} q$, we interpret this as 'p is a homotopy between p and q, and so on.



The slogan to remember here is that, in type theory, *equality is homotopy*. This is a good place to finally make make clear the distinction between traditional and *synthetic* homotopy theory.

- **Traditional homotopy theory**: the study of paths between points (and paths between these paths, and so on) in topological spaces.
- **Synthetic homotopy theory**: the study of the identity type in dependent type theory.

Since equality in type theory *is* homotopy under the types-as-spaces interpretation, you see why these two things really are the same (although, admittedly, it might not be clear yet what equality has to do with homotopy equivalences, but that will be clarified later when we introduce the univalence axiom).

Nomenclature. From now on, we'll almost always refer to elements of identity types as *paths* rather than *proofs*.

Paths in type theory

If the type $x = y$ really is to be thought of as paths from x to y, we ought to be able to define path composition and path reversal. How? The answer comes from the most fundamental properties of the identity type. Let's first briefly recall the J-rule or, as we will call it from now on, **path induction**.

Fact (Path Induction)**.** Let $x : A$ and $B : (y : A) \times (x = y) \to \mathsf{Type}$. In order to construct a (possibly dependent) function

$$f : ((y, p) : (y : A) \times (x = y)) \to B(y, p)$$

it suffices to define

$$f(x, \mathrm{refl}_x) : B(x, \mathrm{refl}_x)$$

Informally: *In order to prove something about $x, y : A$ and $p : x = y$ – where y is an arbitrary point – it suffices to do so when $y \equiv x$ and $p \equiv \mathrm{refl}_x$.*

Note that, since the identity type is symmetric, it's also fine if x is an arbitrary element – as long x and y aren't both fixed points points or depend on each other, the principle holds.

Remark. The following identity types are **not** OK to apply path induction on:

- path types where both endpoints are fixed elements rather than variables, e.g. $0 = 1$, $0 = f(x)$, and $\mathbb{N} = \mathbb{N}$, and
- path types where both endpoints have a term in common, e.g. $x = x$ and $x = f(x)$.

Despite these limitations, path induction is still incredibly powerful. Perhaps most importantly, it can be used to prove the most crucial properties of the identity type:

Lemma. *The following properties hold for the identity type:*

(a) If $x = y$ and $y = z$, then $x = z$

(b) If $x = y$, then $y = x$,

(c) For every x, we have $x = x$.

Proof. Let's construct (a) explicitly. Fix $x, z : A$. We define the type $B : (y : A) \times (x = y) \to \mathsf{Type}$ by

$$B(y, p) :\equiv (y = z \to x = z)$$

We wish to construct a dependent function into B. By path induction, it's enough to construct an element

$$t : \underbrace{B(x, \mathrm{refl}_x)}_{x=z \to x=z}$$

Hence, we simply need to provide a function of type $x = z \to x = z$, so we can define $t :\equiv \mathsf{id}_{x=z}$.

Let's now do (b), but this time in the informal style that homotopy type-theorists *actually* write path induction proofs in. We're given a path $p : x = y$ and wish to define $p^{-1} : y = x$. By path induction, it suffices to do so when $y \equiv x$ and $p \equiv \mathrm{refl}_x$. In other words, it suffices to define

$$\mathrm{refl}_x^{-1} : x = x$$

We define $\mathrm{refl}_x^{-1} :\equiv \mathrm{refl}_x$, and we are done.

For (c), we don't need to do anything: the identity type is, by design, equipped with a path $\mathrm{refl}_x : x = x$. □

There's another way of interpreting this lemma: it's defining the basic path operations:

Lemma.

(a) There is an operation

$$\cdot : (x = y) \times (y = z) \to x = z$$

(path composition)

(b) There is an operation $(-)^{-1} : x = y \to y = z$ *(path reversal)*

(c) There is a path $\mathrm{refl}_x : x = x$ *(the constant path)*

The above operations satisfy the usual laws.

Lemma. *In what follows, let* $p : x = y$, $q : y = z$, *and* $r : z = w$. *We have*

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

Proof. Let's do (a), as the rest of the entries are proved in an almost identical manner. We use path induction on p. This allows us to assume that $y \equiv x$ and $p \equiv \mathrm{refl}_x$. Hence, we only have to show that

$$\mathrm{refl}_x^{-1} \cdot \mathrm{refl}_x = \mathrm{refl}_x \tag{1}$$

By we *defined* $\mathrm{refl}_x^{-1} :\equiv \mathrm{refl}_x$, so the left-hand side above is *exactly* $\mathrm{refl}_x \cdot \mathrm{refl}_x$. However, we also *defined* $\mathrm{refl}_x \cdot (-) :\equiv \mathsf{id}$, and so $\mathrm{refl}_x \cdot \mathrm{refl}_x$ is *exactly* refl_x. Hence (1) is literally just

$$\mathrm{refl}_x = \mathrm{refl}_x$$

which of course holds (by $\mathrm{refl}_{\mathrm{refl}_x}$, to be precise). □

Notice that we can write = rather than ∼ above – this is because there is no distinction between these two notions in type theory.

Restricting path composition/inversion to the case when all paths are loops, we obtain a 'group structure' on $\Omega(X, x) :\equiv (x = x)$. The neutral element is given by refl_x. The primary goal of this course is to show that the loop space of the circle is equal to the integers.

Formally, we like to think of Ω as an operation on *pointed types*:

Definition. A **pointed type** is a pair $(X, \star_X) : \underbrace{(X : \mathsf{Type}) \times X}_{\mathsf{Type}_\star}$ where $X : \mathsf{Type}$ and $\star_X : X$.

Definition. A **pointed function** is a pair $(f; \star_X) : \underbrace{(f : X \to Y) \times (f(\star_X) = \star_Y)}_{(X,\star_X)\to_\star(Y,\star_Y)}$

Notation.

- When there is no risk for confusion, we often simply write $X : \mathsf{Type}_\star$ and $f : X \to_\star Y$, i.e. we leave basepoints and proofs of pointedness implicit.
- We always write $\star_X : X$ for basepoints and $\star_f : f(\star_X) = \star_Y$ for pointedness proofs.

Remark. The identity function is always a pointed function by $\mathsf{refl}_{\star_A} : \mathsf{id}(\star_A) = \star_A$. It is also easy to see that pointed functions are composable, but this requires the notion of *function application*, which will only be introduced on the next page. Nonetheless, it's a good exercise to try to define composition of pointed functions directly here.

Let's restate the definition of loop spaces in terms of pointed types.

Definition. We define the **loop space** of a pointed types X by

$$\Omega(X) := (\star_X = \star_X)$$

This type is itself pointed by $\mathsf{refl}_{\star_X}$. Note that we're again leaving the basepoint implicit, simply writing $\Omega(X)$ rather than $\Omega(X, \star_X)$.

Remark. $\Omega : \mathsf{Type}_\star \to \mathsf{Type}_\star$ can be iterated. This gives a notion of nth loop space.

Lemma. *Ω is functorial, i.e. given a pointed map $f : A \to_\star B$, there is a pointed map $\Omega(f) : \Omega(A) \to_\star \Omega(B)$ s.t.*

- $\Omega(\mathsf{id}_A) = \mathsf{id}_{\Omega(A)}$
- $\Omega(f \circ g) = \Omega(f) \circ \Omega(g)$

We'll leave the proof of this as an exercise. Note that you'll probably have to learn about function application (more on this soon) first.

n-types

Some material is still missing before we'll have enough to prove that $\Omega(S^1) \simeq \mathbb{Z}$. One of the missing concept is that of an *n-type* – roughly, a type that is 'uninteresting (homotopically speaking)' from dimension $(n + 2)$ and above (yes, '+2' – we start indexing from $n \geq -2$...).

The most 'uninteresting' are the *contractible types*:

Definition (Contractible types)**.** A type A is said to be contractible (or a (-2)-type) if

- it has an element $\star_A : X$, and
- for every other element $x : X$, we have that $x = \star_A$

Examples:

- the unit type (i.e. the inductive type $\mathbb{1}$ with a single constructor $\star_{\mathbb{1}} : \mathbb{1}$, and
- the singleton type $(x : X) \times (x = y)$ (c.f. exercise 4). This is essentially the content of the rule/principle of path induction.

Second in the hierarchy are the *propositions*:

Definition (Proposition)**.** A type A is said to be a proposition (or a (-1)-type) if for any $x, y : A$, we have that $x = y$.

Examples:

- any contractible type, and
- the empty type $\bot$.

After this, we have the *sets*.

Definition (Sets)**.** A type A is said to be a set (or a 0-type) if it satisfies the principle of **Uniqueness of Identity Proofs (UIP)**, i.e. if for any $x, y : A$, and $p, q : x = y$, we have that we have that $p = q$.

- Any contractible type and any proposition
- Any 'discrete' type: $\mathbb{Z}$, $\mathbb{N}$, and so on. We won't introduce the machinery to actually prove this in this course, so you'll have to take my word for it.

Remark. In general, we say that a type A is an n-type for $n \geq 0$ if for all $x, y : A$, we have that the type of paths $x = y$ is an $(n-1)$-type. It's a good sanity check to unfold what this actually means in the case when $n = 0$, and see that it agrees with the above definition of 0-types/sets. We will, however, not need n-types for higher n than 0 in this course.

Exercises

1. (Egbert, Ex 5.1) Show that path inversion distributes over path composition, i.e. that for $p : x = y$ and $q : y = z$, we have
$$(p \cdot q)^{-1} = q^{-1} \cdot p^{-1}$$
2. (Egbert, Ex 5.2) Given $p : x = y$, $q : y = z$ and $r : x = z$. Construct functions
$$\mathsf{inv\text{-}con}(p, q, r) : p \cdot q = r \to q = p^{-1} \cdot r$$
$$\mathsf{con\text{-}inv}(p, q, r) : p \cdot q = r \to p = r \cdot q^{-1}$$

3. Given a dependent type $B : A \to \mathsf{Type}$, points $x, y : A$ and a path $p : x = y$, define the transport function

$$\mathsf{transport}^B_p : B(x) \to B(y)$$

4. Let $x : A$. Prove that the **singleton type** $(y : X) \times (x = y)$ is contractible.

5. **(Towards a) characterisation of equality in Σ-types**. Let $(x_0, y_0), (x_1, y_1) : (x : A) \times B(x)$. Construct a function

$$((p : x_0 = y_0) \times \mathsf{transport}^B_p(y_0) = y_1) \to \big((x_0, y_0) =_{(x:A)\times B(x)} (x_1, y_1)\big)$$

Lecture 2: univalence, dependent paths, and the loop space of the circle

Univalence

Univalence states, very simplified, that if two types are 'equivalent', then they are equal. To make this formal, we need to specify what we mean by 'equivalent'.

Intuitively, two types X and Y are equivalent if they're in 'bijection' – i.e. if there are maps $f : X \to Y$ and $g : Y \to X$ s.t. $f \circ g = \mathsf{id}$ and $g \circ f = \mathsf{id}$ (recall the definition of homotopy equivalence in traditional homotopy theory). For all proofs in this course, this intuition suffices. However, for technical reasons that I'll explain later, this is not the appropriate definition. To arrive at a better definition, we'll need to define *fibres*:

Definition. The **fibre** of a function $f : X \to Y$ over a point $y : Y$ is the type

$$\mathrm{fib}_f(y) := (x : X) \times (f(x) = y)$$

Definition. A function $f : X \to Y$ is an **equivalence** if $\mathrm{fib}_f(y)$ is contractible for all y.

We define the type of equivalences between two types X and Y by

$$(X \simeq Y) :\equiv (f : X \to Y) \times \mathrm{isEquiv}(f)$$

It is a good exercise to show that $\mathrm{id}_X : X \to X$ is an equivalence. We can also invert and compose equivalences, although proving this will be significantly easier after you've read the following remark:

Remark. Why don't we use $\underbrace{(g : Y \to X) \times (g \circ f = \mathrm{id}) \times (f \circ g = \mathrm{id})}_{:\equiv \mathrm{qinv}(f)}$ as definition? The issue with this definition is that it is *not a proposition*, so there may be many different proofs of it for a given function f. This makes it inappropriate to use. In particular, it would make the univalence axiom contradictory. However, it turns out that $\mathrm{qinv}(f)$ and $\mathrm{isEquiv}(f)$ both imply each other (i.e. they are *logically equivalent*), so we can just pretend that these definitions are the same.

That's certainly what I do when I construct (most) equivalences. For this reason, I will often talk about the 'inverse' of an equivalence. It's worth noting, however, that when X and Y are sets, the two notions do, in fact, coincide completely.

Axiom (Univalence)**.** *The canonical map* $X =_{\mathsf{Type}} Y \to (X \simeq Y)$ *is an equivalence. In particular we get a map* $\mathsf{ua} : X \simeq Y \to X = Y$

Above, the 'canonical map' is the map $\mathsf{transport}^{Y \mapsto X \simeq Y}_{(-)}(\mathsf{id}_X) : X = Y \to X \simeq Y$. (c.f. transport from Lecture 1, Exercise 3). If you prefer, you can also construct it explicitly by applying path induction to the input – this works because it's of type $X = Y$.

Univalence has some interesting consequences:

Fact. All contractible types are equal to the unit type. Indeed, if X is contractible, we can show that the unique map $X \to 1$ is an equivalence, and hence $X = 1$.

Fact. Univalence contradicts UIP (c.f. Exercise 3).

Fact. Univalence implies **function extensionality**, i.e. the statement that if $(((x\,y : X) \to f(x) = g(x))$ then $f = g$

From now on, we will always assume (and use) univalence/function extensionality, often without comment.

Function application and dependent paths

OK – I said that the goal of this course was to show that $\Omega(S^1) \simeq \mathbb{Z}$. So far we've seen how to define $\Omega(X)$ and we've seen the univalence axiom, which will be crucially used for actually computing $\Omega(X)$. We have not, however, seen how to define any interesting concrete space that we can actually instantiate X with. In particular, we haven't defined S^1. This is the role of *higher inductive types* (HITs), the second and last addition we make to MLTT to get HoTT. In order to introduce these in a meaningful way, we'll need to quickly introduce the notion of *dependent paths* and *(dependent) function application*.

When I write function application, I mean the following principle/construction:

- Given $f : X \to Y$ and $p : x = y$, we have $\mathrm{ap}_f(p) : f(x) = f(y)$ by path induction on p.

From a logical point of view, this principle says that equality is preserved by functions – makes sense. From a topological point of view, it says that all functions preserve paths. That is (roughly): all functions are continuous! It's worth recalling tha laws governing function application, as they will be needed for some of the exercises:

Lemma. *Given* $f : X \to Y$, $g : Y \to Z$, $p : x = y$, *and* $q : y = z$, *we have:*

- $\mathrm{ap}_f(p \cdot q) = \mathrm{ap}_f(p) \cdot \mathrm{ap}_f(q)$

- $\mathrm{ap}_{g \circ f}(p) = \mathrm{ap}_g(\mathrm{ap}_f(p))$
- $\mathrm{ap}_{\mathsf{id}}(p) = p$
- $\mathrm{ap}_{\mathsf{const}}(p) = \mathrm{refl}$

Proof. After path induction on p, all statements become trivial. □

This notion of function application is somewhat lacking: it only applies to plain function but we're doing *dependent* type theory – so what about dependent functions? That is, what should this principle say about a dependent function $f : (x : X) \to B(x)$? We can't write $\mathrm{ap}_f(p) : f(x) = f(y)$ because $f(x) : B(x)$ and $f(y) : B(y)$, so this equality doesn't type check! But $B(x)$ and $B(y)$ are equal types (by p), so we should still be able to express the idea that $f(x)$ and $f(y)$ are 'equal modulo p'. To make sense of this, we introduce *dependent paths*:

Definition (Dependent paths)**.** Given a dependent type $B : X \to \mathsf{Type}$, a path $p : x_0 = x_1$, and elements $b_0 : B(x_0)$ and $b : B(x_1)$, define

$$(b_0 =_p^B b_1) :\equiv (\mathrm{transport}_p^B(b_0) = b_1)$$

Note that the above definition coincides *exactly* with the usual identity type whenever p is refl. With this, we can define *dependent* function application:

- Given $f : X \to Y$ and $p : x = y$, we have $\mathrm{apd}_f : (p : x = y) \to f(x) =_p^B f(y)$ (by path induction on p)

Higher inductive types and the circle

In Loïc's course, we saw some *inductive* types, i.e types defined by a (possibly recursive/inductive) definition of their points and equipped with 'built-in' recursion/induction principles. Examples include:

- The natural numbers, i.e. the inductive type $\mathbb{N}$ defined by
  - A point $0 : \mathbb{N}$
  - For every $n : \mathbb{N}$, a point $\mathsf{suc}(n) : \mathbb{N}$.
- The unit type, i.e. the inductive type $\mathbb{1}$ defined by
  - A point $\star : \mathbb{1}$
- The empty type, i.e. the inductive type $\bot$ with no constructors.

Higher inductive types (HITs) are inductive types that also allow us to specify *path constructors*, effectively gluing together points. You can think of this as a way to to introducing quotients to type theory. For instance, we can define $\mathbb{Z}/2\mathbb{Z}$ as the HIT defined by

- Points $[n] : \mathbb{Z}/2\mathbb{Z}$ for all $n : \mathbb{N}$
- Paths $\mathsf{quot}_n : [n] = [n+2]$ for all $n : \mathbb{N}$

HITs come with recursion/induction principle. In the case of $\mathbb{Z}/2\mathbb{Z}$, these will essentially just say that (possibly dependent) functions out of $\mathbb{Z}/2\mathbb{Z}$ correspond to (possibly dependent) functions out of $\mathbb{N}$ that respect the equalities $[n] = [n+2]$.

Saying that HITs are just there to introduce quotients is a bit too simplistic, however. For our purposes, they are probably even more important as a way to introduce spaces. The canonical example of a HIT that captures a 'nice' topological space is S^1, the circle, and this will be our working example of a HIT for the rest of the course.

Definition. The **circle**, S^1, is the HIT generated by

- A point $\star_{S^1} : S^1$
- A path $\mathsf{loop} : \star_{S^1} = \star_{S^1}$

Draw it

In order to be able to reason about S^1, we need to know how to define functions out of it. That is, we need to state its recursion/induction rules. Let's start with recursion:

Recursion: A map $f : S^1 \to X$ is defined by the following data:

- $x : X$ (i.e. we define $f(\mathsf{base}) :\equiv x$)
- $p : x =_X x$ (i.e. we define $\mathsf{ap}_f(\mathsf{loop}) :\equiv p$)

The rule says that f defined this way satisfies $f(\star_{S^1}) := x$ and $\mathsf{ap}_f(\mathsf{loop}) = p$.

Example. Classically, we can define the 'doubling' map $d : S^1 \to S^1$ by $d(e^{i\pi x}) := e^{2i\pi x}$. In HoTT, it is easily define using the recursion principle:

$$\begin{aligned} d(\star_{S^1}) &:\equiv \star_{S^1} \\ \mathsf{ap}_d(\mathsf{loop}) &:\equiv \mathsf{loop} \cdot \mathsf{loop} \end{aligned}$$

Similarly, we can define the inversion map $(-)^{-}S^1 \to S^1$ (i.e. $e^{i\pi x} \mapsto e^{-i\pi x}$) by

$$\begin{aligned} \star_{S^1}^{-1} &:\equiv \star_{S^1} \\ \mathsf{ap}_{(-)^{-1}}(\mathsf{loop}) &:\equiv \mathsf{loop}^{-1} \end{aligned}$$

A nice way of stating the recursion principle is to say that the evaluation map

$$\begin{aligned} (S^1 \to X) &\to (x : X) \times (x = x) \\ f &\mapsto (f(\star_{S^1}), \mathsf{ap}_f(\mathsf{loop})) \end{aligned}$$

is an equivalence. For instance, the doubling map above corresponds to the pair $(\star_{S^1}, \text{loop} \cdot \text{loop}) : (x : S^1) \times (x = x)$. This is nice, because it's immediately generalisable to the dependent case: for $B : S^1 \to \mathsf{Type}$, the evaluation map

$$\begin{aligned}((x : S^1) \to B(x)) &\to (x : X) \times (x =^B_{\text{loop}} x) \\ f &\mapsto (f(\star_{S^1}), \text{apd}_f(\text{loop}))\end{aligned}$$

is an equivalence. Explicitly:

Induction: A dependent function $f : (x : S^1) \to B(x)$ is determined by the following data

- $x : B(\star_{S^1})$ (i.e. we define $f(\text{base}) :\equiv x$)
- $p : x =^B_{\text{loop}} x$ (i.e. we define $\text{apd}_f(\text{loop}) :\equiv p$)

This principle can be used by prove e.g. that for all $x : S^1$, we have that $d(x^{-1}) = d(x)^{-1}$; in other words, we can use it to construct a dependent function of type

$$(x : S^1) \to d(x^{-1}) = d(x)^{-1}$$

but in the interest of time, we'll have to leave this for the exercises following this lecture.

The fundamental group of S^1

Finally: let's show that $\Omega S^1 \simeq \mathbb{Z}$. We'll use the following machinery to streamline the proof.

Definition. A fibre sequence (of pointed types) $F \to E \xrightarrow{B}$ is a sequence where $F \simeq \text{fib}_f(\star_B)$.

Fact. We always have fibre sequences:

$$B(\star_A) \to (a : A) \times B(a) \to A$$

and

$$\Omega A \to 1 \to A$$

In the above, if $(a : A) \times B(a)$ is contractible (i.e. equivalent to $\mathbb{1}$, the unit type), we would get that $\Omega A \simeq B(\star_A)$. Hence, we can show that $\Omega S^1 \simeq \mathbb{Z}$ by constructing $B : S^1 \to \mathsf{Type}$ s.t. $B(\star_{S^1}) = \mathbb{Z}$ and s.t. $(x : S^1) \times B(x)$ is contractible. This is precisely the plan for the proof.

Defining B We define B by S^1-elimination:

$$\begin{aligned}B : S^1 &\to \mathsf{Type} \\ B(*) &:= \mathbb{Z} \\ \text{ap}_B(\text{loop}) &:= \mathsf{ua}(x \mapsto x + 1)\end{aligned}$$

Fact: We have $\mathrm{transport}^B_{\mathrm{loop}}(x) = x + 1$ and $\mathrm{transport}^B_{-\mathrm{loop}}(x) = x - 1$. This can be shown using *full* statement of the univalence axiom (and the fact that B was constructed using univalence). If we can show that $(x : S^1) \times B(x)$ is contractible, we're done. We need to check two things:

1. It has a point: there is a canonical point of $(x : S^1) \times B(x)$, namely $(\star_{S^1}, 0)$
Write this up more carefully:

2. This point is unique: Given any other two points $x : S^1$ and $y : B(x)$, we need to show that $(\star_{S^1}, 0) =_{(x:S^1)\times B(x)} (x, y)$.

- By analysing what paths between pairs look like (c.f. Exercise 5 from the previous lecture), this corresponds to providing two pieces of data, namely $p : (x : S^1)(r : B(x)) \to \star_{S^1} = x$ and $q : (x : S^1)(r : B(x)) \to 0 =^R_{p(x,r)} r$. We leave q as an exercise but make the following remark:
  - Note that q takes input in S^1 and output in a rather complicated type which we can show is a proposition. It is a fact (that we leave as a direct but somewhat advanced exercise) that in order to define a (possibly dependent) function out of S^1, it suffices to do so over the basepoint. In other words, if we can just define $q(\mathrm{base})$, we're done. We won't have time to do this here, but it's a good (advanced) exercise.

We construct p by S^1-elimination:

$$p(\star_{S^1}) : (r : \mathbb{Z}) \to \star_{S^1} = \star_{S^1}$$

$$p(\star_{S^1})(r) :\equiv \mathrm{loop}^r$$

For the loop case, need to show

$$(r \mapsto \mathrm{loop}^r) =^{x\mapsto B(x)\to\star_{S^1}=x}_{\mathrm{loop}} (r \mapsto \mathrm{loop}^r)$$

More explicitly, this means that for all $r : \mathbb{Z}$, we need a path

$$\mathrm{transport}^{x\mapsto B(x)\to\star_{S^1}=x}_{\mathrm{loop}}(r \mapsto \mathrm{loop}^r)(r) = \mathrm{loop}^r$$

By using the appropriate transport lemmas (advanced exercise: deduce what this lemma should say and how to prove it), we can show the the LHS is the same thing as:

$$\mathrm{transport}^{x\mapsto\star_{S^1}=x}_{\mathrm{loop}}(\mathrm{loop}^{\mathrm{transport}^R_{\mathrm{loop}^{-1}}(r)}).$$

We compute

$$\mathrm{transport}^B_{\mathrm{loop}^{-1}}(r) = r - 1.$$

So

$$\mathrm{transport}^{x \mapsto \star_{S^1} = x}_{\mathrm{loop}}(\mathrm{loop}^{r-1}) = \mathrm{loop}^{r-1} \cdot \mathrm{loop} = \mathrm{loop}^{r}$$

where the first equality again follows from a more general lemma about transports (which one?) and we are done. This proves that $\Omega S^1 \simeq \mathbb{Z}$ (with inverse given by $\mathrm{loop}^{(-)}$).

Todo: write conclusion Fix references

Exercises

Part 1: Equivalences and univalence

1. Prove the following basic equivalences. Note that square brackets are used to increase readability and have no separate meaning from regular parentheses.

   - $A \times A' \simeq A' \times A$
   - $(a : A) \times [(b : B(a)) \times C(a, b)] \simeq [(a, b) : (a : A) \times B(a)] \times C(a, b)$ where $B : A \to \mathsf{Type}$ and $C : (a : A) \to B(a) \to \mathsf{Type}$
   - If $B(a) \simeq B'(a)$ for all $a : A$, then $(a : A) \times B(a) \simeq (a : A) \times B'(a)$
   - If $e : A' \simeq A$ and $B : A \to \mathsf{Type}$, then $(a : A') \times B(e(a)) \simeq (a : A) \times B(a)$
     - **Hint:** constructing the inverse of this equivalence directly is rather cumbersome. You may want to use univalence in order to assume that $e \equiv \mathsf{ua}^{-1}(p)$ for $p : A' = A$ and use path induction.

2. Let A be a pointed type. Show that the fibre of the map $\lambda x \,.\, \star_A :\equiv 1 \to A$ is equivalent to ΩA

3. Recall, UIP is the principle that for any type A, elements $x, y : A$ and proofs $p, q : x = y$, we have that $p = q$. Show that UIP fails if we assume univalence.
   Hint: *Consider the identity type $\mathbb{2} = \mathbb{2}$, where $\mathbb{2}$ denotes the $\mathbb{2}$-element type (i.e. the booleans). You may assume that there are two elements $0, 1 : \mathbb{2}$ s.t. $0 \neq 1$.*

4. Given two pointed types A and B, we define a *pointed* equivalence by

   $$A \simeq_\star B :\equiv (e : A \simeq B) \times (e(\star_A) = \star_B)$$

   Show that $(A \simeq_\star B) \simeq (A = B)$

Part 2: The circle and other HITs

For some of these exercises, you may want to use the following special case of the induction principle for S^1: Given two functions $f, g : S^1 \to A$ we have that $f = g$ iff we can construct

- a path $p : f(\text{base}) = g(\text{base})$
- a higher path $p \cdot \text{ap}_g(\text{loop}) = \text{ap}_f(\text{loop}) \cdot p$

1. ($\star$) give a formal proof of the above principle (using the induction principle for S^1)
2. Show that $(x^{-1})^{-1} = x$ for all $x : S^1$. Conclude that the inversion map $(-)^{-1} : S^1 \to S^1$ is an equivalence.
   You'll need to recall the properties of ap **for this one.**
3. Show that $d(x^{-1}) = (d(x))^{-1}$ for all $x : S^1$. Here, d denote the 'doubling map' that we defined during the lecture.
4. Define a new HIT describing $\mathbb{Z}$ as two copies of $\mathbb{N}$ with their 0s glued together.
5. It's annoying to have to define new HITs for each space we're interested in capturing. In practice most spaces are defined as an instance of the following HIT. Given two maps $f : A \to B$ and $g : A \to C$, we define their pushout (often denoted $B \sqcup_A C$) as the HIT generated by
   - a function $\mathsf{inl} : B \to B \sqcup_A C$
   - a function $\mathsf{inr} : C \to B \sqcup_A C$
   - a family of paths $\mathsf{push} : (a : A) \to \mathsf{inl}(f(a)) = \mathsf{inr}(g(a))$

   Describe the recursion and induction principles for $B \sqcup_A C$.
6. We define the suspension of a type by $\Sigma(A) :\equiv \mathbb{1} \sqcup_A \mathbb{1}$.
   (a) Show that $\Sigma(\bot) \simeq \mathbb{2}$
   (b) ($\star$) Let $\widehat{S}^1 :\equiv \Sigma(\mathbb{2})$. Show that $\widehat{S}^1 \simeq S^1$. Deduce that $\Omega(\widehat{S}^1) \simeq \mathbb{Z}$

Bibliography

Anel, Mathieu, Georg Biedermann, Eric Finster, and André Joyal (2020). 'A generalized Blakers-Massey theorem'. In: *J. Topol.* 13.4, pp. 1521–1553. DOI: 10.1112/topo.12163.

Bezem, Marc, Ulrik Buchholtz, Pierre Cagne, Bjørn Ian Dundas, and Daniel R. Grayson (Apr. 6, 2024). *Symmetry*. https://github.com/UniMath/SymmetryBook. Commit: 994b4f1.

Brunerie, Guillaume (2016). 'On the homotopy groups of spheres in homotopy type theory'. PhD thesis. arXiv: 1606.05916.

— (2017). *The Steenrod squares in homotopy type theory*. TYPES 2017 extended abstract.

— (Aug. 2019). 'The James Construction and $\pi_4(\mathbb{S}^3)$ in Homotopy Type Theory'. In: *J. Autom. Reason.* 63.2, pp. 255–284. DOI: 10.1007/s10817-018-9468-2.

Buchholtz, Ulrik, J. Daniel Christensen, Jarl G. Taxerås Flaten, and Egbert Rijke (2023). *Central H-spaces and banded types*. arXiv: 2301.02636 [math.AT].

Buchholtz, Ulrik, Floris van Doorn, and Egbert Rijke (2018). 'Higher Groups in Homotopy Type Theory'. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '18. Oxford, United Kingdom: Association for Computing Machinery, pp. 205–214. DOI: 10.1145/3209108.3209150. arXiv: 1802.04315.

Buchholtz, Ulrik and Kuen-Bang Hou (Favonia) (June 2020). 'Cellular Cohomology in Homotopy Type Theory'. In: *Logical Methods in Computer Science* Volume 16, Issue 2. DOI: 10.23638/LMCS-16(2:7)2020.

Buchholtz, Ulrik and Egbert Rijke (2017). 'The real projective spaces in homotopy type theory'. In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2017)*. New York, NY, USA: IEEE, pp. 1–8. DOI: 10.1109/LICS.2017.8005146. arXiv: 1704.05770.

— (Nov. 2018). 'The Cayley-Dickson Construction in Homotopy Type Theory'. In: *Higher Structures* 2.1, pp. 30–41. DOI: 10.21136/hs.2018.02.

— (2023). 'The long exact sequence of homotopy n-groups'. In: *Mathematical Structures in Computer Science* 33.8, pp. 679–687. DOI: 10.1017/S0960129523000038. arXiv: 1912.08696.

Cagne, Pierre, Ulrik Buchholtz, Nicolai Kraus, and Marc Bezem (2024). *On symmetries of spheres in univalent foundations*. arXiv: 2401.15037 [cs.LO].

Christensen, J Daniel and Luis Scoccola (July 2023). 'The Hurewicz theorem in homotopy type theory'. In: *Algebraic & Geometric Topology* 23.5, pp. 2107–2140. DOI: 10.2140/agt.2023.23.2107.

Christensen, J. Daniel, Morgan Opie, Egbert Rijke, and Luis Scoccola (Feb. 2020). 'Localization in Homotopy Type Theory'. In: *Higher Structures* 4.1, pp. 1–32. DOI: 10.21136/hs.2020.01.

Christensen, J. Daniel and Egbert Rijke (2022). 'Characterizations of modalities and lex modalities'. In: *Journal of Pure and Applied Algebra* 226.3, p. 106848. DOI: 10.1016/j.jpaa.2021.106848.

Doorn, Floris van (2018). 'On the Formalization of Higher Inductive Types and Synthetic Homotopy Theory'. PhD thesis. Carnegie Mellon University. arXiv: 1808.10690 [math.AT].

Graham, Robert (2018). *Synthetic Homology in Homotopy Type Theory*. arXiv: 1706.01540 [math.LO].

Hou (Favonia), Kuen-Bang, Eric Finster, Daniel R. Licata, and Peter LeFanu Lumsdaine (2016). 'A Mechanization of the Blakers-Massey Connectivity Theorem in Homotopy Type Theory'. In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '16. New York, NY, USA: Association for Computing Machinery, pp. 565–574. ISBN: 9781450343916. DOI: 10.1145/2933575.2934545.

Hou (Favonia), Kuen-Bang and Robert Harper (2018). '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, 11:1–11:16. DOI: 10.4230/LIPIcs.TYPES.2016.11.

Hou (Favonia), Kuen-Bang and Michael Shulman (2016). 'The Seifert-van Kampen Theorem in Homotopy Type Theory'. In: *25th EACSL Annual Conference on Computer Science Logic (CSL 2016). Leibniz International Proceedings in Informatics (LIPIcs)*. Vol. 62. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 22:1–22:16. DOI: 10.4230/LIPICS.CSL.2016.22.

Kraus, Nicolai and Thorsten Altenkirch (2018). 'Free Higher Groups in Homotopy Type Theory'. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '18. Oxford, United Kingdom: Association for Computing Machinery, pp. 599–608. ISBN: 9781450355834. DOI: 10.1145/3209108.3209183.

Kraus, Nicolai and Jakob von Raumer (2021). 'Path spaces of higher inductive types in homotopy type theory'. In: *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '19. Vancouver, Canada: IEEE Press. DOI: 10.5555/3470152.3470159.

Lamiaux, Thomas, Axel Ljungström, and Anders Mörtberg (2023). 'Computing Cohomology Rings in Cubical Agda'. In: *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2023. New York, NY, USA: Association for Computing Machinery, pp. 239–252. DOI: 10.1145/3573105.3575677.

Licata, Daniel R. and Eric Finster (2014). 'Eilenberg-MacLane spaces in homotopy type theory'. In: *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. CSL-LICS '14. Vienna, Austria: Association for Computing Machinery. DOI: 10.1145/2603088.2603153.

Licata, Daniel R. and Michael Shulman (2013). 'Calculating the Fundamental Group of the Circle in Homotopy Type Theory'. In: *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '13. USA: IEEE Computer Society, pp. 223–232. DOI: 10.1109/LICS.2013.28.

Ljungström, Axel and Anders Mörtberg (June 2023). 'Formalizing $\pi_4(S^3) \simeq \mathbb{Z}/2\mathbb{Z}$ and Computing a Brunerie Number in Cubical Agda'. In: *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE. DOI: 10.1109/lics56636.2023.10175833.

Lumsdaine, Peter LeFanu and Daniel R. Licata (2012). *Freudenthal Suspension Theorem*. Agda formalization. URL: https://github.com/dlicata335/hott-agda/commits/master/homotopy/Freudenthal.agda.

Myers, David Jaz (July 2022). 'Good Fibrations through the Modal Prism'. In: *Higher Structures* 6.1, pp. 212–255. DOI: 10.21136/hs.2022.04.

Myers, David Jaz and Mitchell Riley (2023). *Commuting Cohesions*. arXiv: 2301.13780 [math.CT].

Rijke, Egbert (2022). 'Introduction to Homotopy Type Theory'. Book draft; version of 8 April. URL: https://raw.githubusercontent.com/martinescardo/HoTTEST-Summer-School/main/HoTT/hott-intro.pdf.

Rijke, Egbert, Michael Shulman, and Bas Spitters (2020). 'Modalities in homotopy type theory'. In: *Log. Methods Comput. Sci.* 16.1, Paper No. 2, 79.

Scoccola, Luis (2020). 'Nilpotent types and fracture squares in homotopy type theory'. In: *Mathematical Structures in Computer Science* 30.5, pp. 511–544. DOI: 10.1017/S0960129520000146.

Shulman, Michael (2018). 'Brouwer's fixed-point theorem in real-cohesive homotopy type theory'. In: *Math. Structures Comput. Sci.* 28.6, pp. 856–941. DOI: 10.1017/S0960129517000147.

— (2021). 'Homotopy type theory: the logic of space'. In: *New spaces in mathematics—formal and conceptual reflections*. Cambridge Univ. Press, Cambridge, pp. 322–403.

Sojakova, Kristina and G. A. Kavvos (2022). 'Syllepsis in Homotopy Type Theory'. In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '22. Haifa, Israel: Association for Computing Machinery. DOI: 10.1145/3531130.3533347.

Swan, Andrew W (Jan. 2022). 'On the Nielsen-Schreier Theorem in Homotopy Type Theory'. In: *Logical Methods in Computer Science* Volume 18, Issue 1. DOI: 10.46298/lmcs-18(1:18)2022.

The HoTT Library (n.d.). a Coq library of formalized proofs, available at https://github.com/HoTT/HoTT.

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

Voevodsky, Vladimir, Benedikt Ahrens, Daniel Grayson, et al. (n.d.). *UniMath — a computer-checked library of univalent mathematics*. available at http://UniMath.org.

Wärn, David (Sept. 2023). 'Eilenberg–Maclane spaces and stabilisation in homotopy type theory'. In: *Journal of Homotopy and Related Structures* 18.2–3, pp. 357–368. DOI: 10.1007/s40062-023-00330-5.

— (2024). *Path spaces of pushouts*. arXiv: 2402.12339 [math.AT].