# Homotopy Type Theory

Brandon Shapiro
Cornell Logic Seminar

Homotopy type theory is, at its core, an alternative foundational system for mathematics built to more naturally describe mathematical objects exhibiting "higher dimensional structure" or relating to each other via weaker notions of equivalence than identity or isomorphism. These sorts of rich structures have been studied for decades by mathematicians including homotopy theorists and algebraic geometers, but using complicated techniques that suggest set theoretic and topological foundations are not ideal for expressing and reasoning about such concepts.

In homotopy type theory, the fundamental objects are *types*, which seamlessly play the role of both spaces and logical propositions. The axioms of these types encode both homotopical and logical structure, allowing them to serve as both theory and metatheory in interesting ways. For instance, given two types $A$ and $B$, there is a type which can be regarded as containing either homotopy equivalences between $A$ and $B$ or *proofs* that $A$ and $B$ are homotopy equivalent.

The main thematic difference between type theory and classical mathematics is that propositions in the former are regarded as types, which come equipped with all of the axiomatic structure of the types they refer to. A proof of a proposition is, in type theory, a first class mathematical object, which can be compared and contrasted with other proofs of the same proposition. This perspective is most interesting when applied to equality of elements within a type, where spatial structure arises by regarding proofs that two elements are equal as paths between those elements.

Even in classical mathematics however, homotopy type theory provides a syntax for reasoning about spaces in a particularly intuitive way that translates into proofs in a variety of different settings at once. Homotopy theorists often bounce between point-set, combinatorial, and algebraic presentations of abstract spaces. Homotopy type theory uses algebraic descriptions of spaces to state and prove results across a variety of combinatorial and point-set presentations.

## 1. Type Theory

This section follows chapter 1 in The Book. The axioms given here are consistent (relative to ZFC, more on this later), but redundant as many of the type constructors in

this section could be defined in terms of the others. We do modify the order in which different type constructors are presented from the book, which begins with more complicated constructors in order to give more detailed descriptions of those that follow than we will provide in these notes, so we instead present the standard constructors in order of increasing complication.

### 1.1. Judgments and Types

Types are, like sets, defined by a list of axioms. These axioms suffice to describe both the objects of study (types) and the methods for reasoning about them (also types). The fundamental syntactic atom of type theory is, for $A$ a type, the *judgment*

$$a : A$$

where a judgment is not a proposition but rather a definition of the *element* $a$ of type $A$. Everything in type theory belongs to a type, and it is usually helpful to assume that every element belongs to a unique type, though we will have to stray from this to define dependent types. $a : A$ defines (or in some cases assumes) the element $a$ as having type $A$, and can also be seen as a proof that the type $A$ has an element. When we regard propositions as types, $a : A$ will be taken as providing a proof of $A$. As a judgment and not a proposition, however, $a : A$ is not something that can be proven or disproven; it simply specifies an element $a$ with type $A$.

We also have another judgment, written $a \equiv b : A$, for when $a$ and $b$, both elements of type $A$, are *definitionally equal*. Definitional equality is basically a metatheoretical convenience for expressing when two things are the same by definition and can be substituted for one another in type theoretic statements without using any type theory axioms. This differs from set theory where "$a = b$" is a proposition that can be proven true or false by axioms and deduction rules, whereas in type theory we have $a \equiv b$ only when one or the other is introduced as being the same as the first. Axioms can, however, be used to obtain $p : a = b$ an element of the type $a = b$, the main innovation of this foundational system.

Since everything has a type, this includes the types themselves. We will define new types $A$ by the judgment $A : U$, where $U$ is called the *universe type* whose elements are types. Most of the following axioms for type theory can be interpreted as providing an inductive definition of $U$. The type of $U$ is typically taken to be a larger universe $U_1$ fitting into a hierarchy of successive universes. While it is simpler to consider each type as belonging to a unique universe in the hierarchy, we will want to consider functions of

the form $A \to U$ for $A : U$, so in these cases it becomes convenient to allow types in $U$ to also be treated as types in $U_1$. The intricacies of these conventions are however beyond the scope of these notes, where we will freely move between different universes in the hierarchy without further comment.

### 1.2. Functions and Implication

So far we have only introduced syntax indicating that types are "things of type $U$ which may have elements," which does not tell us much about how to use them. Like sets, however, there can also be functions between types, defined by the following axioms:

**Axiom.**

$$\frac{A : U \quad B : U}{A \to B : U}\,1.1 \qquad \frac{a : A \vdash f(a) : B}{f : A \to B}\,1.2 \qquad \frac{f : A \to B \quad a : A}{f(a) : B}\,1.3$$

In axioms 1.2 and 1.3 we also assume $A, B : U$ (shorthand for $A : U \quad B : U$), but we will omit these assumptions when they are clear from context. The "cut"-notation here, meaning some number of premises placed above a line with a conclusion below the line, is a standard scheme for specifying axioms.

The first axiom (1.1) defines a new type $A \to B$ for any types $A$ and $B$. The third axiom (1.3) provides that an element of the type $A \to B$ can be "applied" (like a function) to an element in $A$ to get an element of $B$. The second axiom (1.2) uses a "turnstile" $\vdash$, and has the following meaning: if for an arbitrary element $a : A$ one can form an expression $f(a)$ that describes an element of $B$, then this assignment specifies a function $f : A \to B$. However, while it is important to be able to construct new functions, we will rarely use this axiom as most types are defined not by specifying all of their elements, but rather by describing the data necessary to give functions out of them, as we will see in the axioms that follow.

Furthermore, using a slight abuse of notation, the function $f$ obtained from axiom 1.2 when applied to an element $a : A$ as in axiom 1.3 yields the same expression $f(a)$ used in defining $f$. Similarly, given $f : A \to B$, the function constructed from axiom 1.2 by the assignment $g(a) :\equiv f(a)$ yields a function $g$ with $g \equiv f$. These interactions amount to the *uniqueness principle* for functions, which asserts that they are definitionally determined by their values on all elements of the domain type $A$. Later on, we will introduce an additional axiom that provides a similar statement using equality types.

For example, given an element $b : B$, we can define $const_b : A \to B$ by $a : A \vdash b : B$,

with $f(a) \equiv b$ for all $a : A$. If $A$ were a type with an operation $+$, which we will discuss later, we could similarly define a function $f$ by $f(a) :\equiv (a + a) + a$. There is also always an identity function $id_A : A \to A$ defined by $f(a) \equiv a$. Given $f : A \to B$ and $g : B \to C$ we can define their composite $gf : A \to C$ by $gf(a) \equiv g(f(a))$.

Like all of our basic type constructors, $A \to B$ has a logical interpretation as well. As suggested by the notation, if $A$ and $B$ are regarded as propositions, $A \to B$ corresponds to "$A$ implies $B$." We can interpret $a : A$ as a proof of $A$, and a function $f : A \to B$ takes that proof and provides a proof $f(b) : B$ of $B$, just like implication in classical logic. Axiom 1.3 corresponds to *modus ponens* and axiom 1.2 corresponds to proving $A \to B$ by deducing $B$ from $A$ constructively (as in, explicitly using the given proof of $A$ to prove $B$).

### 1.3. Unit and Empty Types as Truth Values

Before defining several more ways of constructing new types from existing ones, we provide two new types as base cases. The first is the *unit type* 1, characterized by the property of having an element which behaves as if it is unique. We will later describe the precise sense in which this element is unique.

**Axiom.**

$$\frac{}{1 : U}\,2.1 \qquad \frac{}{\star : 1}\,2.2 \qquad \frac{A : U \quad a : A}{\bar{a} : 1 \to A \qquad \bar{a}(\star) \equiv a}\,2.3$$

As with the axioms for function types, these axioms follow the scheme of defining a new type, specifying how to construct elements in it (and accordingly functions into it), and specifying how to construct functions out of it. In this case, the type 1 has an element $\star$ by axiom 2.2, and by axiom 2.3 functions out of 1 can be constructed by only defining their value on that element, which is like asserting that $\star$ is the unique element of 1.

Axiom 2.3 also provides a "computation rule" that we did not include for function types for simplicity of notation, though we did discuss it following the function type axioms. This rule describes how the constructed functions out of 1 act on the element $\star$. This could be derived from axiom 1.2 if we knew that $\star$ is the unique element of 1, but it takes the first conclusion of axiom 2.3 to know anything like this, and even then it does not explicitly guarantee that $\star$ is unique. We will see that in fact, we can only show syntactically that $\star$ is unique up to *propositional equality*, but not definitional equality. Semantically however, the type 1 is often modeled by a mathematical object in which $\star$ is the unique element.

**Axiom.**

$$\frac{}{0 : U}3.1 \qquad \frac{A : U}{\iota_A : 0 \to A}3.3$$

0 is called the *empty type*, and as suggested by its lack of any specified elements (which would be axiom 3.2) and existence of a function out of it into any type, it has no elements. As such there is no need for a computation rule in axiom 3.3.

From the perspective of logic, 1 and 0 correspond to the propositions $True$ and $False$. 1 has not just a proof but a canonical proof (in the sense that one can construct a proof that any proof of 1 is equal to $\star$), and having no elements 0 has no proof.

As further evidence for the validity of this correspondence, note that for any type $A$ there are canonical functions $!_A : A \to 1$, obtained from axiom 1.2 by setting $!_A(a) = \star$, as well as $\iota_A : 0 \to A$. These functions correspond with the classical implications of anything by $False$ and of $True$ by anything. There is a function $1 \to A$ precisely when $A$ has an element (aka proof), and a function $A \to 0$ only when $A$ has no elements. Accordingly, $\sim A :\equiv A \to 0$ is the type theoretic analogue of negation.

This is where type theory is first seen to be naturally *constructive*. With logical propositions replaced by types, many classical axioms/theorems of logic arise naturally from the axioms, such as modus ponens above or propositions like $A \to 1$. These classical notions are constructive in the sense that if given a proof (sometimes called a *witness*) of each of the premises, like $f : A \to B$ and $a : A$, one can construct a proof of the conclusion, like $f(a) : B$.

A famously nonconstructive proposition in classical logic is that $\sim\sim A \to A$. The converse is constructive, as given $a : A$ we can construct an element of $(A \to 0) \to 0$ by sending each $f$ in $(A \to 0$ to $f(a) : 0$. But given a function $(A \to 0) \to 0$ there is no way to extract an element of $A$.

Part of the richness of this constructive approach to logic is that propositions are not simply true or false, but are rather collections of all their possible proofs, some of which can be proven equal but potentially in different ways, some of which can be proven equal, and so on. This nested structure of proofs of equality of proofs of equality of (...) endows a type with the higher dimensional structure of a space, which we discuss further below.

### 1.4. Product and Sum Types

Now that we have a basic sense of what type theory axioms look like and how to work with them, we introduce two new ways of constructing new types, which correspond to product and coproduct in spaces, or conjunction and disjunction (and,or) in logic.

**Axiom.**

$$\frac{A : U \quad B : U}{A \times B : U}4.1 \qquad \frac{a : A \quad b : B}{(a,b) : A \times B}4.2 \qquad \frac{f : A \to (B \to C)}{\bar{f} : A \times B \to C \qquad \bar{f}((a,b)) \equiv f(a)(b)}4.3$$

Any two types have a product $A \times B$, whose elements are contstructed from an element of $A$ along with an element of $B$. Axiom 4.3, the *elimination rule* for products, describes functions out of products in terms of functions on each component. Classically in sets (and sufficiently nice spaces) a function out of a product is the same as a function from the first component into functions out of the second component, corresponding via the computation rule in the second conclusion of axiom 4.3. From the functions $A \to (B \to A)$ sending $a : A$ to $const_a$ and $A \to (B \to B)$ sending $a : A$ to $id_B$ for all $a$, we get functions $pr_1 : A \times B \to A$ with $pr_1((a,b)) \equiv a$ and $pr_2 : A \times B \to B$ with $pr_2((a,b)) \equiv b$. This type corresponds to conjunction ($\wedge$) in logic.

**Axiom.**

$$\frac{A : U \quad B : U}{A + B : U}5.1 \qquad \frac{a : A}{inl(a) : A + B}5.2a \qquad \frac{b : B}{inr(b) : A + B}5.2b$$

$$\frac{f : A \to C \quad g : B \to C}{\langle f, g\rangle : A + B \to C \quad \langle f, g\rangle(inl(a)) \equiv f(a) \quad \langle f, g\rangle(inr(b)) \equiv g(b)}5.3$$

Elements of the sum type $A + B$ are constructed *either* as an element of $A$ or an element of $B$, and these are all of the elements in the sense that a function $A + B \to C$ can be specified on only the elements of $A$ and $B$. This is also an example of a type construction with multiple different constructors for elements. It is evident from these axioms that $+$ of types behaves like disjunction ($\vee$) in logic, but with the constructive caveat that a proof of $A + B$ does in fact specify either a proof of $A$ or a proof of $B$.

### 1.5. Dependent Type Theory

So far, we have constructed enough types to do constructive propositional logic: we have $0, 1, \to, \wedge, \vee, \sim$ all behaving as they should in a constructive setting, atoms of the form $A : U$, and proofs of the form $a : A$, with appropriate rules for forming proofs of or from

the various type constructions. But propositional logic isn't strong enough to do most classical mathematics. Dependent type theory adds additional constructions resembling those of first order logic, but with types again assuming multiple roles, including that of the relations and the indexing set. Specifically, dependent products/sums look like iterated products/sums indexed over a type, and correspond to universal and existential quantification.

This is the setting, mentioned at the beginning of the section, where the universe must be treated at the same level as types it contains. In first order logic, universal and existential quantification assert that for a family of propositions which depend on a variable, respectively all or at least one of those propositions is true. We replace those propositions with types, depending on an element of an indexing type, so that a family of types is described by a function $B : A \to U$, called a *type family*. For each $a : A$, $B(x)$ is a type, and the dependent product and sum describe respectively a family of elements of all those types and an element of a specific one of those types, as follows.

**Axiom.**

$$\frac{A : U \quad B : A \to U}{\prod_{x:A} B(x) : U}\,6.1 \qquad \frac{x : A \vdash b_x : B(x)}{b : \prod_{x:A} B(x)}\,6.2 \qquad \frac{b : \prod_{x:A} B(x) \quad a : A}{b_a : B(a)}\,6.3$$

An element of $\prod_{x:A} B(x)$ thus amounts to an element of $B(x)$ for all $x : A$. Logically, an element of $\prod_{x:A} B(x)$ can be seen as a proof that "for all $x$ in $A$, B(x) holds" via a proof of each $B(x)$.

Note that these axioms look a lot like those for function types, including the abuse of notation indicating that the operations between types in axioms 6.2 and 6.3 are inverse to each other. This analogy becomes precise when the type family $B : A \to U$ is constant at a single type ($B$), in which case an element of $\prod_{a:A} B$ is simply a function $A \to B$. Dependent product types are often thought of as function types where the codomain of the function is allowed to depend on the element it is applied to in the domain. In dependent type theory, all of the axioms for functions out of newly constructed types (like axioms $n.3$ above) are replaced with axioms for such dependent functions out of the type, which look similar and we won't write them out here. However, we state below the dependent version of that axiom for dependent sums, and in the next section the "path induction" axiom for identity types cannot be stated without dependent products.

**Axiom.**

$$\frac{A : U \quad B : A \to U}{\sum_{x:A} B(x) : U}\,7.1 \qquad \frac{a : A \quad b : B(a)}{(a, b) : \sum_{x:A} B(x)}\,7.2 \qquad \frac{C : (\sum_{x:A} B(x)) \to U \quad x : A, y : B(x) \vdash z_{x,y} : C((x, y))}{z : \prod_{p:\sum_A B} C(p)}\,7.3$$

An element of $\Sigma_{x:A} B(x)$ is a pair $(a, b)$ with $a : A$ and $b : B(a)$. Logically this is interpreted as a constructive variant of existential quantification, as it contains proofs of $B(a)$ for some $a$, but the specific choice of $a$ must be specified, more than the mere assertion that there is such an $a$. A nonconstructive version of this that behaves more like classical existential quantification can be defined using propositional truncation, which uses higher inductive types.

Where dependent products behave like functions with the codomain type dependent on elements in the domain, dependent sums behave like pairs with the type of the second component dependent on elements in the type of the first component. When $B$ is a constant type family, an element of $\Sigma_{x:A} B(x)$ amounts to a pair in $A \times B$. Axiom 7.3 constructs dependent functions out of $\Sigma_{x:A} B(x)$, but choosing $C$ to be constant reduces it to constructing ordinary functions out of $\Sigma_{x:A} B(x)$. For instance, if $C$ is constant at $A$, assigning $z_{x,y} :\equiv x : A$ defines the first projection $pr_1 : \Sigma_{x:A} B(x) \to A$. If $C$ sends $(x, y)$ to $B(x)$, then assigning $z_{x,y} :\equiv y : B(x)$ defines the second projection $pr_2 : \Pi_{\Sigma_{x:A} B(x)} B(x)$, a dependent function.

By now, we have the tools to fully imitate constructive first order logic using our type theory. However, we have yet to introduce any interesting propositions to prove. In the next section we introduce identity types and in the following section the type of natural numbers, after which we can construct elements of types like

$$\prod_{n:\mathbb{N}} \sum_{k:\mathbb{N}} (n = k + k) + (n = k + k + 1)$$

where the $+$ used in $k + k$ is defined for natural numbers and the $+$ between parenthesized blocks is the sum of types. An element of this type would be a proof that every natural number is either even or odd.

### 1.6. Axioms So Far

For convenience we include here all of the axioms given thus far:

**Axiom.**

$$\frac{A : U \quad B : U}{A \to B : U}1.1 \qquad \frac{a : A \vdash f(a) : B}{f : A \to B}1.2 \qquad \frac{f : A \to B \quad a : A}{f(a) : B}1.3$$

**Axiom.**

$$\frac{}{1 : U}2.1 \qquad \frac{}{\star : 1}2.2 \qquad \frac{A : U \quad a : A}{\bar{a} : 1 \to A \qquad \bar{a}(\star) \equiv a}2.3$$

**Axiom.**

$$\frac{}{0 : U}3.1 \qquad \frac{A : U}{\iota_A : 0 \to A}3.3$$

**Axiom.**

$$\frac{A : U \quad B : U}{A \times B : U}4.1 \qquad \frac{a : A \quad b : B}{(a, b) : A \times B}4.2 \qquad \frac{f : A \to (B \to C)}{\bar{f} : A \times B \to C \qquad \bar{f}((a, b)) \equiv f(a)(b)}4.3$$

**Axiom.**

$$\frac{A : U \quad B : U}{A + B : U}5.1 \qquad \frac{a : A}{inl(a) : A + B}5.2a \qquad \frac{b : B}{inr(b) : A + B}5.2b$$

$$\frac{f : A \to C \quad g : B \to C}{\langle f, g\rangle : A + B \to C \quad \langle f, g\rangle(inl(a)) \equiv f(a) \quad \langle f, g\rangle(inr(b)) \equiv g(b)}5.3$$

**Axiom.**

$$\frac{A : U \quad B : A \to U}{\Pi_{x:A}\, B(x) : U}6.1 \qquad \frac{x : A \vdash b_x : B(x)}{b : \Pi_{x:A}\, B(x)}6.2 \qquad \frac{b : \Pi_{x:A}\, B(x) \quad a : A}{b_a : B(a)}6.3$$

**Axiom.**

$$\frac{A : U \quad B : A \to U}{\Sigma_{x:A}\, B(x) : U}7.1 \qquad \frac{a : A \quad b : B(a)}{(a, b) : \Sigma_{x:A}\, B(x)}7.2 \qquad \frac{C : (\Sigma_{x:A}\, B(x)) \to U \quad x : A, y : B(x) \vdash z_{x,y} : C((x, y))}{z : \Pi_{p:\Sigma_A\, B}\, C(p)}7.3$$

## 2. Propositional Equality

This section follows (roughly) 1.12-2.12 in The Book.

Thus far, all of the type theory we have covered could be modeled by sets, where the types are interpreted as sets and all of the constructions in the previous section are treated as their set theoretic analogues. The most basic propositions about those sets are equalities $x = y$ between two sets or elements, and in type theory we treat propositions as types. From the perspective of sets, either $x$ and $y$ are equal or they are not, so a type $x = y$ should either be empty or have a canonical element. Homotopy type theory arises from dropping the assumption that elements of the type $x = y$ have to be unique,

while imposing conditions that allow elements of this *identity type* between $x$ and $y$ to be interpreted as paths from $x$ to $y$, making the ambient type behave like a space.

Many important properties of identity types can be proven from the axioms below, but the essential aspects of these types are that only two elements of the same type can be equal, every element of a type is equal to itself in a canonical way, and these *reflexivity paths* generate all other paths in a precise sense.

**Axiom.**

$$\frac{A : U \quad a : A \quad b : A}{a =_A b : U}\,8.1 \qquad \frac{a : A}{refl_A : a =_A a}\,8.2 \qquad \frac{C : (\Sigma_{a,b:A}\, a = b) \to U \quad c : \Pi_{a:A}\, C(a, a, refl_a)}{d : \Pi_{a,b:A}\, \Pi_{p:a=b}\, C(a, b, p) \quad d(a, a, refl_a) = c(a)}\,8.3$$

Here $\Pi_{a,b:A}$ is convenient notation for $\Pi_{a:A}\, \Pi_{b:A}$, and we write $a = b$ for $a =_A b$ when the ambient type is clear from context. In axiom 8.3, $C$ is a family of types in $U$ consisting of a type $C(a, b, p)$ for every path $p$ from $a$ to $b$, and we note that a family $(\Sigma_{a,b:A}\, a = b) \to U$ is the same as a family $\Pi_{a,b:A}(a = b \to U)$.

The axiom is called the *path induction principle*, which says that given a proposition indexed by all paths in $A$, if it can be proven for just the reflexivity paths $refl_a$ for all $a : A$, then it is proven for all paths. This only works when $C$ varies over all paths in $A$, and would not work for a single identity type in isolation like $C : a = b \to U$. This may seem too strong; after all, in a space most paths are not the constant paths that the reflexivities corresponds to. But in a space $X$, the path space $X^I$ is homotopy equivalent to $X$, as any path can be contracted back to the constant path at its first endpoint.

Path induction lets us do proofs by "induction on paths", where any path variable universally quantified over (including quantifying over its endpoints) can be assumed to be a reflexivity path without loss of generality. We do several of these sorts of proofs below. In these proofs, we use the special case of the induction principle that says to provide an element of $\Pi_{a,b:A}\, a = b \to C(a, b)$ it suffices to give an element of $\Pi_{a:A}\, C(a, a)$.

### 2.1. Groupoid Structure

One would hope that when interpreting equality between elements as a type, it still exhibits familial properties, like the axioms of an equivalence relation. Already from axiom 8.2 we have reflexivity: $refl_{(-)} : \Pi_{a:A}\, a =_A a$. Symmetry and transitivity are great examples of how to use path induction.

**Proposition 1.** *(Symmetry)*

$$(-)^{-1} : \prod_{a,b:A} a = b \to b = a$$

*Proof.* By path induction, we can assume $p : a = b$ is $refl_a$ so the proof reduces to a proof of $\prod_{a:A} a = a$, which is given by $refl_{(-)}$. □

**Proposition 2.** *(Transitivity)*

$$(-;-) : \prod_{a,b,c:A} a = b \to (b = c \to a = c)$$

*Proof.* As the type $a = b$ does not depend on $c$ we can rewrite this type as $\prod_{a,b:A} a = b \to \prod_{c:A}(b = c \to a = c)$, then by path induction assume that $p : a = b$ is $refl_a$. This reduces the proof to $\prod_{a:A} \prod_{c:A} a = c \to a = c$, at which point we could use the identity on $a = c$ for all $a, c$. However, to illustrate a common technique involving path induction, we instead apply it again to assume that $q : a \to c$ is $refl_a$, reducing the proof to $\prod_{a:A} a = a$, provided again by $refl_{(-)}$. In other words, using path induction twice it suffices to define $refl_a; refl_a :\equiv refl_a$. □

These proofs construct composites and inverses of paths in a type $A$ generated by trivial composites and inverses of the reflexivity paths. This shows that *propositional equality* forms an equivalence relation on the elements of a type (if they were to form a set), but also provides algebraic choices for the composites and inverses. These operations, and the identity paths provided by $refl$, satisfy the axioms of a groupoid *up to propositional equality.*

**Definition 3.** A groupoid consists of

- a collection of *objects*
- for any two such objects $a$ and $b$, a set $Hom_(a, b)$ of *morphisms* (also called *arrows* or *maps*)
- an "identity" morphism $id_a \in Hom(a, a)$ for each object $a$
- a "composition" operation $(-;-) : Hom(a, b) \times Hom(b, c) \to Hom(a, c)$ for all triples of objects $a, b, c$
- an "inverse" operation $(-)^{-1} : Hom(a, b) \to Hom(b, a)$

satisfying the equations $id_a; f = f = f; id_b$, $f; f^{-1} = id_a$, and $f^{-1}; f = id_b$ for $f \in Hom(a, b)$, along with $(f; g); h = f; (g; h)$ for $f \in Hom(a, b), g \in Hom(b, c), h \in Hom(c, d)$ (in these equations "=" refers to equality in sets).

For intuition, a groupoid can be thought of in a number of different manners:

- A group where the elements are arrows pointing from one object to another and only successive arrows can be multiplied
- A category in which every morphism has an inverse
- An equivalence relation where two elements can be related in more than one way and reflexivity/symmetry/transitivity are constructive: for instance given a way of relating $a$ and $b$ symmetry picks a particular way to relate $b$ and $a$. The equations in the definitions are to make sure these witnesses to reflexivity/symmetry/transitivity are "coherent" in some sense.

So how do types relate to groupoids? Groupoids are defined above using set theory, so until we are able to talk about sets in our type theory there is no reason to expect an exact correspondence, but types do behave very much like "weak higher groupoids". For instance, we have the following equations:

**Proposition 4.**

$$unit_l : \prod_{a,b:A} \prod_{p:a=b} refl_a; p = p \qquad unit_r : \prod_{a,b:A} \prod_{p:a=b} p; refl_b = p$$

$$inv_l : \prod_{a,b:A} \prod_{p:a=b} p; p^{-1} = refl_a \qquad inv_r : \prod_{a,b:A} \prod_{p:a=b} p^{-1}; p = refl_b$$

$$assoc : \prod_{a,b,c,d:A} \prod_{p:a=b} \prod_{q:b=c} \prod_{r:c=d} (p; q); r = p; (q; r)$$

*Proof.* All of these proofs use path induction to assume $p$ is $refl_a$ (as are $q$ and $r$ by successive appeals to path induction). Given that assumption, both sides of all five equations above are definitionally equal to $refl_a$, so it suffices to give $refl_{refl_a} : refl_a = refl_a$. For instance in the last equation, $assoc$ is induced by $refl_{refl_a} : (refl_a; refl_a); refl_a = refl_a; (refl_a; refl_a)$, which is correctly typed as both $(refl_a; refl_a); refl_a \equiv refl_a$ and $refl_a; (refl_a; refl_a) \equiv refl_a$. □

While these elements can be interpreted as proofs of the corresponding equations in the definition of a groupoid, they only show the equations hold under propositional

equality; in the spatial interpretation, this amounts to a "path of paths" from $(p;q);r$ to $p;(q;r)$ rather than an actual equality of the two. This is consistent with how composition of paths works in topology, and suggests that the groupoid laws for 1-dimensional paths only hold up to 2-dimensional paths as witnesses. Those 2-dimensional paths then satisfy additional "coherence equations" again only up to higher dimensional paths induced from path induction by the constant $n$-dimensional path $refl_{refl_{\cdots refl_a}}$, and the process continues infinitely.

To complicate matters even further, the types $a = b$ for $a, b : A$ have their own identity types $p =_{a=b} q$ of paths between the paths $p$ and $q$ (themselves paths from $a$ to $b$ for fixed $a, b$). These higher paths also have composition and inverses and all of the groupoid structure proven for general types above. The structure consisting of a collection of *cells* in each dimension along with identity, composition, and inverse operations along with higher dimensional cells witnessing the groupoid laws (which need not hold strictly) is called a *weak $\omega$-groupoid*. The axioms for types presented here suggests that a type and all of its nested identity types behave like a weak $\omega$ groupoid, which is classically considered an algebraic model for the homotopy information of a space (via the "homotopy hypothesis" of Grothendieck).

Functions between types preserve the groupoid structure, again by path induction:

**Proposition 5.**

$$(-)_* : \prod_{f:A\to B}\prod_{a,b:A}(a =_A b) \to (f(a) =_B f(b))$$

$$preserves\_refl : \prod_{f:A\to B}\prod_{a:A} f_*(refl_a) = refl_{f(a)}$$

$$preserves\_comp : \prod_{f:A\to B}\prod_{a,b,c:A}\prod_{p:a=_A b}\prod_{q:b=_A c} f_*(p;q) =_B f_*(p);f_*(q)$$

$$preserves\_inv : \prod_{f:A\to B}\prod_{a,b:A}\prod_{p:a=_A b} f_*(p^{-1}) = f_*(p)^{-1}$$

*Proof.* For the action of $f$ on paths, by path induction it suffices to define $f_*(refl_a) :\equiv refl_{f(a)}$. This defines $preserves - refl$ without using path induction as the equation it witnesses is a definitional equality so $refl_{refl_a}$ is a complete definition. Path induction then reduces $preserves - comp$ to $refl_{refl_{f(a)}} : refl_{f(a)} = refl_{f(a)};refl_{f(a)}$ and $preserves - inv$ to $refl_{refl_{f(a)}} : refl_{f(a)} = refl^{-1}_{f(a)}$. □

From the perspective of spaces, "preserving the groupoid structure" corresponds to continuity. Functions preserve paths, as well as paths between those paths since the functions $(a =_A b) \to (f(a) =_B f(b))$ also preserve groupoid structure. They also respect the

concatenation and reversal structure of paths. While this is not quite the general definition of topological continuity, for spaces built by gluing together disks in various dimensions (like CW-complexes) it is sufficient, and all of our types correspond to spaces of that sort.

### 2.2. Homotopies, Function Extensioinality, and Univalence

In set theory, two functions $A \to B$ are equal when they agree on every element of $A$. In homotopy type theory, a similar relation can be defined between functions, where now we need a choice of equality $f(x) = g(x)$ for all $x : A$.

**Definition 6.** For functions $f, g : A \to B$, the type of homotopies from $f$ to $g$ is

$$f \sim g :\equiv \prod_{x:A} f(x) =_B g(x)$$

While logically this reads as "for all $x : A$ we have $f$ and $g$ agree on $x$," when equalities are interpreted as paths it resembles an $A$-indexed family of paths in $B$, which is precisely how homotopies are defined between functions of spaces.

This also lets us define when a function is a homotopy equivalence:

**Definition 7.** For $f : A \to B$, we define the type

$$isEquiv(f) :\equiv (\sum_{g:B\to A} fg \sim id_B) \times (\sum_{h:B\to A} hf \sim id_A)$$

An element of $isEquiv$ looks like a pair of functions $g, h : B \to A$ and homotopies $fg \sim id_B$ and $hf \sim id_A$. If $g$ and $h$ were the same function, this would correspond exactly to the standard definition of a homotopy equivalence of spaces. This definition is equivalent however, and is a nicer type for reasons discussed later. The type $isEquiv(f)$ serves as both the proposition that $f$ is a homotopy equivalence, and the type of possible witnesses $g, h$ to $f$ being a homotopy equivalence. We can now define a type of homotopy equivalences from $A$ to $B$.

**Definition 8.** For types $A$ and $B$, the type of homotopy equivalences between them is

$$A \simeq B :\equiv \sum_{f:A\to B} isEquiv(f)$$

An element of $A \simeq B$ consists of a function $f$ and a proof that $f$ is a homotopy equivalence, which consists of the data discussed above.

Homotopy equivalence provides a notion of equivalence for types, which allows us to begin characterizing the identity types in the types arising from the basic constructors in the first section. For instance, the following axiom asserts that two functions are considered equal (by an element of the type $f = g$) precisely when they are homotopic (via a homotopy in $f \sim g$).

**Axiom.** *(Function Extensionality) For types $A, B : U$ and $f, g : A \to B$,*

$$(f =_{A \to B} g) \simeq (f \sim g)$$

Function extensionality imposes that, up to propositional equality, a path between two functions is precisely a homotopy between them, a reasonable notion from the perspective of topology, and that two functions are equal when they agree up to propositional equality on every element, a reasonable notion from the perspective of set theory. The types $f = g$ and $f \sim g$ are here related only by homotopy equivalence, but the next axiom asserts that this is exactly the right way to compare two types.

**Axiom.** *(Univalence) For types $A, B : U$, $(A =_U B) \simeq (A \simeq B)$.*

Univalence is the foundational axiom of homotopy type theory, and makes it so that homotopy equivalent types are equal. This means, among other things, that anything true of a type $A$ is also true of any homotopy equivalent type $B$, so all statements in this type theory are in a sense "up to homotopy." In non-homotopical mathematics, it amounts to something like "isomorphic structures are equal," and similarly suggests that anything definable in the theory can transfer from one structure to any other isomorphic one.

Univalence also tells us that the characterization of identity types in $A \to B$ using homotopy equivalence also holds using equality, which looks like $(f = g) = (f \sim g)$. Additionally, function extensionality can be proven using univalence and the previous axioms.

### 2.3. Equality in Basic Type Constructors

While these first characterizations of identity types required an additional axiom, the remaining type constructors can be characterized as a consequence of the existing axioms. For the type constructors $0, 1, \times, +$, identity types work basically the same way as paths in the corresponding spaces $\emptyset, *, A \times B, A \sqcup B$. Of course, as 0 has no elements it also has no identity types.

**Proposition 9.**

$$\prod_{a,b:1}((a =_1 b) \simeq 1)$$

This says that any two elements of 1 are not only equal, but equal in a canonical way. Furthermore, letting $b$ be $\star$ shows that every element is canonically equal to $\star$, which is the precise sense in which $\star$ is unique, since every other element is canonically equal to it. As a space, this means 1 is *contractible* as there is a family of paths from any element to a fixed basepoint $\star$.

**Proposition 10.** *For $A, B : U$, we have*

$$\prod_{x,y:A\times B}((x =_{A\times B} y) \simeq ((pr_1(x) =_A pr_1(y)) \times (pr_2(x) =_B pr_2(y)))$$

This says that a path between elements of a product amounts to a path between the projections in each component. For pairs $(a, b), (a', b') : A \times B$, a path between the two is completely determined (up to propositional equality) by a path from $a$ to $a'$ and a path from $b$ to $b'$. This is precisely how paths in a product work in spaces, and also demonstrates that $x = (pr_1(x), pr_2(x))$ using reflexive paths, so every element of $A \times B$ is equal to a pair.

**Proposition 11.** *For $a, a' : A$ and $b, b' : B$, we have $(inl(a) =_{A+B} inl(a')) \simeq (a =_A a')$, $(inr(b) =_{A+B} inr(b')) \simeq (b =_B b')$, and $(inl(a) =_{A+B} inr(b)) \simeq 0$.*

This is perhaps less satisfying than the previous characterizations, as it only applies to the constructors and not general elements of $A + B$, which illustrates how types defined by multiple constructors are in many ways more difficult to work with.

For the dependent type constructors we will not discuss their identity types, which are covered in chapter 2 of The Book. However, as in the definitions of these constructors, their identity types behave similarly to function and product types, where dependent functions have a function extensionality axiom implied by univalence, and paths between dependent pairs look like a path in both components, though in the second component the elements related by a path must be modified so as to be in the same type.

## 3. Fancy Type Constructions

The theory of univalent types laid out above emphasizes the ability to interpret types as spaces, but the actual types definable from the axioms so far are not particularly interesting. To remedy this, we informally describe a family of axioms that can be added to the type theory to construct types that model familiar sets and spaces.

### 3.1. Natural Numbers and Inductive Types

In type theory, it is standard to define the natural numbers in the following fashion, somewhat analogous to its definition in set theory.

**Axiom.**

$$\frac{}{\mathbb{N} : U} \qquad \frac{}{0_N : \mathbb{N}} \qquad \frac{}{S : \mathbb{N} \to \mathbb{N}} \qquad \frac{a : A \quad f : \mathbb{N} \to (A \to A)}{g : \mathbb{N} \to A \quad g(0_N) \equiv a \quad g(S(n)) \equiv f(n)(g(n))}$$

This definition of $\mathbb{N}$ fits a pattern of defining types by *induction*, where the type is *generated* by some number of constructors given by specified elements or functions involving a type. In this case, there is an element $0_N$ and a successor function, generating a type with (distinct) elements $0_N, S0_N, SS0_N, ... : \mathbb{N}$. Here what it means to generate a type is not particularly clear, but is made precise by the elimination property, which establishes that given a starting point in $A$ and a means of generating the next term in a sequence in $A$, a function out of $\mathbb{N}$ is determined. This suggests the sense in which the elements $n$ and $Sn$ are distinct and all of the elements of $\mathbb{N}$.

There is a general formulation, discussed in chapter 5 of The Book, of what sorts of constructors are allowed in inductive definitions, which gives rise to a more general axiom to be added to the type theory.

### 3.2. Spaces as Higher Inductive Types

In a type theory with identity types, it is desirable to be able to construct types with control over what their identity types look like. While it is more difficult to formalize than classical induction (which does not involve identity types) and a general scheme for what it looks like is an active research topic, in this section we give examples of what some higher inductive definitions should look like and how they model familiar spaces. All of these definitions could be added as axioms or derived from a general axiom of higher inductive generation, and here we list them as examples. We will sometimes overload notation for elements of these types, but it should not be taken to suggest that any element is actually a member of more than one type.

**Example 12.** We define the interval type $I$ inductively from

$$0_I : I \qquad 1_I : I \qquad i : 0_I =_I 1_I$$

where we have an elimination law given by

$$\frac{a, b : A \quad p : a =_A b}{\bar{p} : I \to A \quad \bar{p}(0_I) \equiv a \quad \bar{p}(1_I) \equiv b \quad \bar{p}_*(i) \equiv p}$$

$I$ is then the "free interval" type, generated by two points and a path between them, with functions out of it picking out the same information in the codomain.

**Example 13.** We define the circle type $S^1$ inductively from

$$base : S^1 \qquad loop : base =_{S^1} base$$

so $S^1$ contains an element $base$ and a loop $loop$ which we have no reason to expect to be the same as $refl_{base}$, as $S^1$ is *freely* generated by these constructors. This is captured by the elimination property

$$\frac{a : A \quad p : a =_A a}{\bar{p} : S^1 \to A \quad \bar{p}(base) \equiv a \quad \bar{p}_*(loop) \equiv p}$$

While $S^1$ should be thought of as a single loop, in fact we have all of $loop, refl_{base}, loop^{-1}, loop; loop, ... : base = base$, so there are in fact many loops in $S^1$ generated by $loop$, and it can be proven that all of those listed are distinct from each other.

**Example 14.** We define the sphere type $S^2$ inductively from

$$base : S^1 \quad loop_2 : refl_{base} =_{base=base} refl_base$$

Here $loop_2$ is a 2-dimensional path from the constant path $refl_{base}$ to itself, which takes the form of a sphere with basepoint $base$. Maps out of $S^2$ correspond to 2-loops $refl_a =_{a=_A a} refl_a$ in a type $A$.

**Example 15.** The disk $D^2$ is defined inductively from

$$base : D^2 \quad loop : base =_{D^2} base \quad disk : loop =_{base=_{D^2} base} refl_{base}$$

which loops like $S^1$ but with a 2-dimensional path to the constant path at $base$ filling in the nontrivial loop. However, as with the 2-sphere, higher inductive types defined explicitly using 2-dimensional paths are more difficult to work with in practice, so there is a benefit to an alternative definition using only 1-dimensional paths (the same can be done for $S^2$). $D^2$ can also be defined inductively from

$$center : D^2 \quad bdry : S^1 \to D^2 \quad filler : \prod_{x:S^1} bdry(x) =_{D^2} center$$

where by continuity of dependent functions, $filler$ provides not just a path from $bdry(base)$ to $center$, but also a continuously varying family of paths to $center$ from around $loop$ (manifesting as a 2-dimensional path, but without needing to specify one in the definition).

These types model spaces, and homotopy theoretic statements about them can be formulated and proven, including calculations of their fundamental groups.

## 4. Logic and Set Theory

In addition to providing foundations in which homotopy theory constructions are more fundamental, homotopy type theory can also model classical logic and set theory, aided by the addition of higher inductive constructions.

### 4.1. Propositional Truncation

While in general types may have many distinct elements, adding proof-relevance to the logical interpretation of types, classical logic can be modeled by restricting to types where proofs are unique up to propositional equality. Where generally types can be viewed as propositions, these types are nothing more than that, and as such are called *mere propositions*.

**Definition 16.** A type $A$ is mere proposition if the following type is inhabited:

$$isProp(A) :\equiv \prod_{a,b:A} a = b$$

$isProp(A)$ asserts that all elements in $A$ are equal to each other, and by continuity of dependent functions (which looks similar to the description of continuity for ordinary functions given above) these paths are *coherent*, in the sense that they vary continuously in $a$ and $b$ and make $A$ into a contractible space when it is nonempty (as in, equivalent to 1).

Mere propositions are meant to behave more faithfully like those in classical logic, and are closed under most of the type constructors ($\to, 0, 1, \times$, $\Pi$ when all $B(x)$ are mere propositions) but not $+$ or $\Sigma$. Therefore, to do classical logic we need a way of turning a general type into a mere proposition. Higher inductive types allow us to perform such a construction.

**Definition 17.** For any type $A$, its *propositional truncation* $|A|$ is defined inductively from:

$$|-| : A \to |A| \qquad contr : \prod_{x,y:A} |x| =_{|A|} |y|$$

That is, $|A|$ contains all the elements of $A$, with coherent paths added between all pairs of elements making it contractible (if inhabited). $|A|$ is always a mere proposition, so classical logic can be imitated with $|A + B|$ and $|\Sigma_{x:A} B(x)|$ playing the roles of $\vee$ and $\exists$ respectively.

While even among mere propositions the system retains some constructive elements, we can add additional axioms to impose the structure of classical logic. For instance the law of excluded middle can be expressed as

$$LEM : \prod_{A:U} (isProp(A) \to (A+ \sim A))$$

which is consistent with existing axioms. Asserting the more general proposition $\Pi_{A:U}(A+ \sim A)$ is, however, inconsistent with univalence, as an element would have to specify a basepoint for all inhabited types, and that basepoint would not generally be preserved by all homotopy equivalences, which contradicts continuity of the dependent function from $U$.

### 4.2. Sets

We can also define a condition on a type for it to be regarded as a set. Among spaces, sets are those which are discrete, having no nontrivial paths. From the perspective of univalence, however, all properties hold only up to homotopy equivalence, so it is not necessary for a set-like type to have no nontrivial paths, but in order to be homotopy equivalent to a discrete type there must be no nontrivial loops (in any dimension).

**Definition 18.** A type $A$ is a set if the following type is inhabited:

$$isSet(A) :\equiv \prod_{a,b:A} \prod_{p,q:A} p = q$$

This immediately eliminates the possibility of nontrivial loops, and higher dimensional loops are avoided by continuity of dependent functions, making each connected component of $A$ contractible. As with propositions, we can *truncate* a type to get a set of its connected components. This is often called 0-truncation, as sets are those types with no nontrivial homotopical structure above dimension 0 (points). From this perspective propositional truncation is (-1)-truncation.

**Definition 19.** For any type $A$, its 0-truncation $|A|_0$ is defined inductively from:

$$|-| : A \to |A|_0 \qquad \prod_{a,b:A} \prod_{p,q:a=_A b} |p| =_{|a|=_{|A|_0}|b|} |q|$$

where $|p|$ is shorthand for $|-|_*(p)$.

This truncation contains all elements of $A$, with each identity type made contractible. As with propositional truncation, nothing is strictly identified as in usual quotient constructions in classical math. Instead, paths are added in, only contracting the type up to homotopy. This is a theme both in homotopy theory and homotopy type theory.

Given this truncation, we can define the *homotopy groups* of types, which allows these groups to be computed in the type theory for types such as those above built to model familiar spaces.

**Definition 20.** For a type $A$ with a specified element $a : A$, its first homotopy group is

$$\pi_1(A, a) :\equiv |\prod_{\gamma:S^1\to A} \gamma(base) = a|_0$$

and higher homotopy groups are defined similarly by replacing $S^1$ with $S^n$.

$\pi_1(A, a)$ is then the set of connected components of the type of loops in $A$ based at $a$, which by function extensionality (ignoring the basepoint condition for simplicity) amounts to looking at loops up to homotopy, just as the fundamental group as defined for topological spaces. Many homotopy groups have been computed in this type theory, including the most basic $\pi_1(S^1, base) = \mathbb{Z}$ with $loop$ the generator under $refl_{base}$, ; and $(-)^{-1}$.

Lastly, as an illustration of the extent to which type theory can model classical mathematics, we can add as an additional axiom an element of the following type, which models the axiom of choice: Let $A$ be a set, $B : A \to U$ with $B(x)$ a set, and $P : (\Sigma_{x:A} B(x)) \to U$ with $P(x, y)$ a mere proposition. Then the axiom of choice amounts to the following.

$$AoC_{A,B,P} : (\prod_{x:A} |\sum_{y:B(x)} P(x, y)|) \to |\sum_{g:\Pi_{x:A} B(x)} \prod_{x:A} P(x, g(x))|$$

That is, given a family of sets indexed by a set $A$ each equipped with a nonempty subset, there exists a dependent function picking out an element of each of those subsets. This would be trivial without the propositional truncations as a proof of nonemptiness of each subset would come equipped with a choice of element, but with them it resembles the classical axiom of choice.

## 5. Models

A detailed discussion of models is beyond the scope of these notes, but we will briefly describe what a model would look like. All of the type theory presented in these notes

has been *syntax*, consisting of strings of symbols with rules for their manipulation. The question "what is a type" has no single answer beyond "anything satisfying these axioms" in some suitable sense.

These axioms are significantly richer than the theories typically studied in model theory, so it would be difficult to model them as a set with functions and relations, especially since types and their elements all have distinct identities, unlike set theory where models can assume all elements are also sets. As such, the machinery for building models of type theory take a different approach, where a model of type theory is a *category* where each type is modeled by some object in that category. The rules for each type construction are then translated into conditions for those objects to satisfy in the category.

For instance, function types behave like exponentials in a category, products and sums like categorical products and coproducts, 0 and 1 like initial and terminal objects, and so on. An informal motto, which has now been proven in various formulations, is that a category that models homotopy type theory has the structure of an $\infty$-topos.

The most significant part of this structure is how identity types are modeled, which relies on the category having a notion of *path object* to model the type $\Sigma_{a,b:A}\, a = b$. In spaces, this would look like $X^I$ where $X$ is the corresponding space to the type $A$ and $I$ is the unit interval.

This structure is formally similar to the structure of a *model category*, which describes a setting that admits constructions from homotopy theory. As such, most models of homotopy type theory are built from the structure of model categories, particularly ones containing combinatorial models of spaces like *simplicial sets*, though there are general conditions that more general model categories can satisfy that allow them to model univalent type theory.

The existence of a model for homotopy type theory with univalence built from ZFC foundations demonstrates that the theory is consistent if ZFC is.

## 6. References

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

2. *The Simplicial Model of Univalent Foundations (after Voevodsky).* Chris Kapulkin and Peter LeFanu Lumsdaine. 2012. https://arxiv.org/abs/1211.2851