OPLSS 2023
INTRODUCTION TO HOTT NOTES

PAVEL KOVALEV, SEAN O'CONNOR, CASSIA TORCZON, FRANK TSAI

Contents

1. Introduction

The fields of Type theory, Set theory, Topos theory, Category theory, Homotopy theory, and Functional programming all matured in the 20th century as their own distinct and influential fields of study. But near the beginning of the 21st century,

Date: July 2023.

more and more of the connections across these fields were recognized, which years later culminated in the field of Homotopy Type Theory (HoTT).

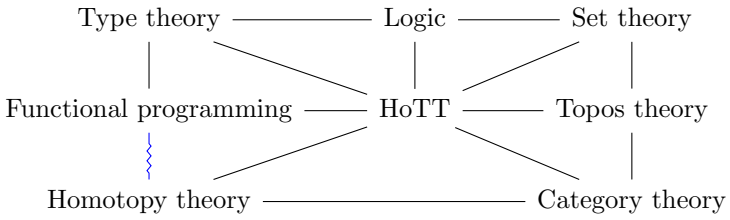

FIGURE 1. Connections across different fields of mathematics and computer science. The blue squiggly line is the surprising connection between homotopy theory and functional programming.

HoTT has strong ties to each of these other fields, and provides a fertile ground for investigating and proving surprising connections between each of them. This lecture series will provide an introduction to this growing field and will help illuminate its relation to these other areas of study. HoTT is built on top of Martin-Löf Type Theory (MLTT), which we introduce in §2.

2. Martin-Löf Type Theory

MLTT consists of 3 basic judgments. An in-depth treatment of judgments can be found in [ML96]. The first judgment expresses that Γ is a valid context.

$$\Gamma \ \mathsf{ctx}$$

The second judgment expresses that T is a type under the context Γ, presupposing that Γ ctx.

$$\Gamma \vdash T \ \mathsf{type}$$

Finally, the last judgment expresses that t is an element of type T under the context Γ, presupposing that Γ ctx and that $\Gamma \vdash T$ type.

$$\Gamma \vdash t : T$$

These three judgments have their corresponding equality judgments. The first judgment expresses that two valid contexts, Γ and Γ', are definitionally equal.

$$\Gamma \doteq \Gamma' \ \mathsf{ctx}$$

The second judgment expresses that two types are definitionally equal under the same context Γ.

$$\Gamma \vdash T \doteq T' \ \mathsf{type}$$

And the last judgment expresses that two terms, t and t', of type T under the context Γ are definitionally equal.

$$\Gamma \vdash t \doteq t' : T$$

For conciseness, we omit the context Γ. For example, we write $\vdash t : T$ for $\Gamma \vdash t : T$, where Γ is an arbitrary context.

Example 2.1 (Some example types in MLTT)**.**

(1) Empty type $\varnothing$.
(2) Unit type $\mathbb{1}$.
(3) The type of Boolean Bool.
(4) The type of natural numbers $\mathbb{N}$.
(5) The types of well-founded trees W.
(5) The types of equalities $=$.
(6) Dependent product types Σ.
(7) Dependent function types Π.
(8) Universes $\mathcal{U}_i$.

MLTT can be extended with additional features. Two possible extensions are summarized in Table 1.

Type Theory	Univalence Axiom	Higher Inductive Types
UTT	✓	✗
HoTT	✓	✓

TABLE 1. Univalent Type Theory (UTT) is MLTT extended with the Univalence Axiom, and HoTT is UTT extended with higher inductive types.

2.1. **Inductive Types.** An inductive type is freely generated by its canonical elements. To define an inductive type in Coq, one can write

```
Inductive T : Type := foo : T | bar : T.
```

The type `T` is freely generated by its two canonical elements, namely `foo` and `bar`. Coq generates an elimination rule for `T` automatically.

In general, to specify a new type in type theory we specify:

(1) how to form new types of this kind via **formation rules**. For example, if A and B are types then $A \to B$ is a type.
(2) how to construct canonical elements of that type via **introduction rules**. For example, a function type has one introduction rule, namely λ-abstraction.
(3) how to use elements of that type via **elimination rules**. For example, a function type has one elimination rule, namely function application.
(4) how elimination rules act on introduction rules via **computation rules**. For example, applying a λ-abstraction $(\lambda x.e)$ to a term e' is definitionally equal to substituting e' for x in e, namely $e[e'/x]$.

See the HoTT book [Uni13].

2.2. **The Booleans.** The type Bool is freely generated by two terms, true and false. To use a Boolean (to construct a dependent function out of Bool), it suffices to specify its behaviors on the canonical terms. The computation rules express that $\mathsf{ind}_{\mathsf{Bool},f,t}(b)$ works as expected when applied to one of the canonical terms.

$$\frac{}{\vdash \mathsf{Bool}\ \mathsf{type}}\ \textsc{Form} \qquad \frac{}{\vdash \mathsf{true} : \mathsf{Bool} \qquad \vdash \mathsf{false} : \mathsf{Bool}}\ \textsc{Intros}$$

$$\frac{b : \mathsf{Bool} \vdash D(b)\ \mathsf{type} \qquad \vdash f : D(\mathsf{false}) \qquad \vdash t : D(\mathsf{true})}{b : \mathsf{Bool} \vdash \mathsf{ind}_{\mathsf{Bool},f,t}(b) : D(b)}\ \textsc{Elim}$$

$$\frac{b : \mathsf{Bool} \vdash D(b)\ \mathsf{type} \qquad \vdash f : D(\mathsf{false}) \qquad \vdash t : D(\mathsf{true})}{\vdash \mathsf{ind}_{\mathsf{Bool},f,t}(\mathsf{false}) \doteq f : D(\mathsf{false}) \qquad \vdash \mathsf{ind}_{\mathsf{Bool},f,t}(\mathsf{true}) \doteq t : D(\mathsf{true})}\ \textsc{Comp}$$

Exercise 2.1. Define a function $\mathsf{not} : \mathsf{Bool} \to \mathsf{Bool}$. Note that $\to$ works as usual.

2.3. **The Natural Numbers.** The type $\mathbb{N}$ is freely generated by 0 and the successor function S. One may recognize that the elimination rule is the usual mathematical induction.

$$\frac{}{\vdash \mathbb{N}\ \mathsf{type}}\ \textsc{Form} \qquad \frac{}{\vdash 0 : \mathbb{N}}\ \textsc{Intros} \qquad \frac{\vdash n : \mathbb{N}}{\vdash S(n) : \mathbb{N}}\ \textsc{Intros}$$

$$\frac{n : \mathbb{N} \vdash D(n)\ \mathsf{type} \qquad \vdash z : D(0) \qquad n : \mathbb{N}, h : D(n) \vdash i : D(S(n))}{n : \mathbb{N} \vdash \mathsf{ind}_{\mathbb{N},z,i}(n) : D(n)}\ \textsc{Elim}$$

$$\frac{n : \mathbb{N} \vdash D(n)\ \mathsf{type} \qquad \vdash z : D(0) \qquad n : \mathbb{N}, h : D(n) \vdash i : D(S(n))}{\mathsf{ind}_{\mathbb{N},z,i}(0) \doteq z : D(0) \qquad n : \mathbb{N} \vdash \mathsf{ind}_{\mathbb{N},z,i}(S(n)) \doteq i[^{\mathsf{ind}_{\mathbb{N},z,i}(n)}/_{h}] : D(S(n))}\ \textsc{Comp}$$

Example 2.2. To construct a function $\mathsf{add} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$, it suffices to construct a term $n : \mathbb{N}, m : \mathbb{N} \vdash e : \mathbb{N}$. To this end, we do induction on m (apply the elimination rule for $\mathbb{N}$), choosing $n : \mathbb{N}, m : \mathbb{N} \vdash \mathbb{N}\ \mathsf{type}$ to be the motive of induction. Now it suffices to produce two terms: $n : \mathbb{N} \vdash z : \mathbb{N}$, corresponding to the base case, and $n : \mathbb{N}, m : \mathbb{N}, h : \mathbb{N} \vdash i : \mathbb{N}$, corresponding to the inductive case. Choosing z to be n and i to be $S(h)$ gives the usual addition function $\lambda n : \mathbb{N}.\ \lambda m : \mathbb{N}.\ \mathsf{ind}_{\mathbb{N},n,S(h)}(m)$ with the following definitional equalities:

- $\mathsf{add}\ n\ 0 \doteq n$, and
- $\mathsf{add}\ n\ S(m) \doteq S(\mathsf{add}\ n\ m)$.

Exercise 2.2. Define $\iota : \mathsf{Bool} \to \mathbb{N}$.

Exercise 2.3. Define $\mathsf{mult} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$.

2.4. **$\sum$-Types.** Given $x : B \vdash E(x)\ \mathsf{type}$ (in Coq: `E (x : B) : UU`), sigma type is freely generated by dependent pairs $\langle b, e\rangle$ where $b : B$ and $e : E(b)$.

$$\frac{x : B \vdash E(x)\ \mathsf{type}}{\vdash \sum_{x:B} E(x)\ \mathsf{type}}\ \textsc{Form} \qquad \frac{\vdash b : B \qquad \vdash e : E(b)}{\vdash \langle b, e\rangle : \sum_{x:B} E(x)}\ \textsc{Intros}$$

$$\frac{z : \sum_{x:B} E(x) \vdash D(z)\ \mathsf{type} \qquad x : B, y : E(x) \vdash d(x,y) : D(\langle x, y\rangle)}{z : \sum_{x:B} E(x) \vdash \mathsf{ind}_{\Sigma,d}(z) : D(z)}\ \textsc{Elim}$$

$$\frac{z : \sum_{x:B} E(x) \vdash D(z)\ \mathsf{type} \qquad x : B, y : E(x) \vdash d(x,y) : D(\langle x, y\rangle)}{b : B, e : E(b) \vdash \mathsf{ind}_{\Sigma,d}(\langle b, e\rangle) \doteq d(b,e) : D(\langle b, e\rangle)}\ \textsc{Comp}$$

We can think of $\sum_{x:B} E(x)$ as the disjoint union of sets indexed by the elements of B. See Figure 2.

Example 2.3. Given $x : \mathsf{Bool} \vdash \mathbb{N}\ \mathsf{type}$, the type $\Sigma_{x:\mathsf{Bool}}\mathbb{N}$ corresponds to the set $\mathbb{N} \sqcup \mathbb{N}$, and $\Pi_{x:\mathsf{Bool}}\mathbb{N}$ corresponds to the set $\mathbb{N} \times \mathbb{N}$ because there is no dependency between x and $\mathbb{N}$ in the Π type.

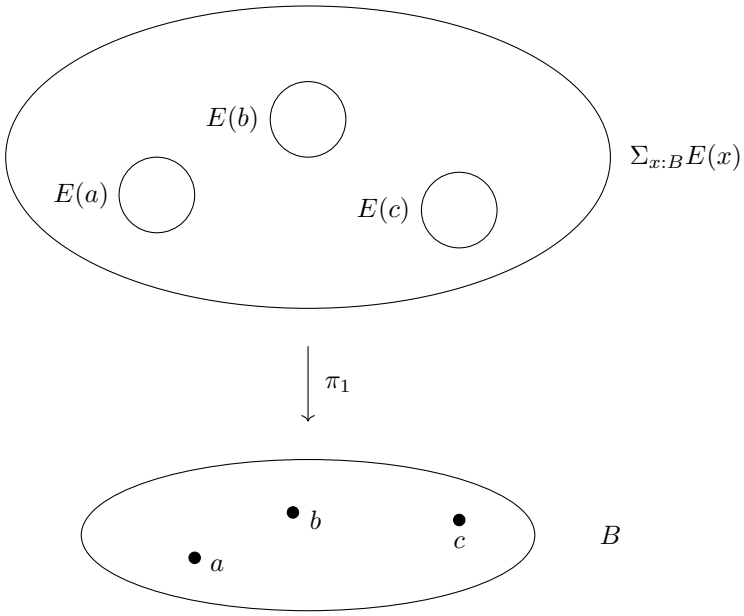

FIGURE 2. A graphical illustration of a $\sum$-type over some type B.

Exercise 2.4. Construct a function $\pi_1 : \sum_{x:B} E(x) \to B$.

Exercise 2.5. Construct a function $\pi_2 : \prod_{s:\Sigma_{x:B}E(x)} E(\pi_1 s)$. This is a dependently typed function that takes a term $s : \sum_{x:B} E(x)$ and returns something of type $E(\pi_1 s)$.

2.5. **Types as Logic, Sets, and Programs.** This section compares previously presented terms in light of the Curry-Howard correspondence, which relates logic to programs, and the Brouwer–Heyting–Kolmogorov interpretation, which provides an explanation of constructive logic. Table 2 summarizes the interpretation of type theory in logic, sets, and programs.

3. Identity Types

In Example 2.2, $\mathsf{add}\ 0\ m \doteq m : \mathbb{N}$ doesn't hold judgmentally because we don't know whether m is 0 or is the successor of some natural number. To "prove" this equality, we need to use the elimination rule for $\mathbb{N}$. However, since $\doteq$ is not a type, it does not make sense to make the judgment $\vdash t : \mathsf{add}\ 0\ m \doteq 0$. We can internalize the existing notion of equalities, namely judgmental equalities ($\doteq$). This yields the identity types, giving us another notion of equalities, **propositional equalities**.

Type formers often internalize existing concepts, i.e., turn them into types. For example, Bool, $\mathbb{N}$, $\varnothing$ and $\mathbb{1}$ internalize existing notions of those things; Σ-types internalize context extensions (Figure 3); Π-types internalize dependent types; and the universe type $\mathcal{U}$ internalizes the notion of something being a type. We will see how the identity types internalize judgmental equalities.

Given a type A and two terms, $a : A$ and $b : A$, we can ask whether a equals b, i.e., we can form the type $a =_A b$. A term $a : A$ is equal to itself; thus the canonical terms of this type family are $r_a : a =_A a$. The elimination rule tells us that to use

Type Theory	Logic	Sets	Programs
Γ ctx	hypotheses	indexing set	names in scope
$\Gamma \vdash T$ type	predicate T on Γ	family T of sets indexed by Γ	program specification using values from Γ
$\Gamma \vdash t : T$	proof of T	elements	program t meeting the specification T
$\mathbb{N}$	-	$\mathbb{N}$	collection of programs with no input that output a natural number
$S + T$ ($\Sigma_{i:\mathsf{Bool}} T_i$)	$\vee$ (disjunction)	$\sqcup$ (disjoint union)	$\vee$ over specifications
$S \times T$ ($\Sigma_{s:S} T$)	$\wedge$ (conjunction)	$\times$ (Cartesian product)	$\wedge$ over specifications
$S \to T$ ($\Pi_{s:S} T$)	$\Rightarrow$ (implication)	T^S (exponential)	turns a program of type S into a program of type T
$\Sigma_{b:B} E(b)$	$\exists b : B, E(b)$ (constructive existential)	$\sqcup_{b:B} E(b)$	specification for producing an element $b : B$ meeting specification $E(b)$
$\Pi_{b:B} E(b)$	$\forall b : B, E(b)$	Π (set of sections)	specification for taking any program $b : B$ and outputting a program matching specification $E(b)$

TABLE 2. Type theory interpreted in logic, sets, and programs.

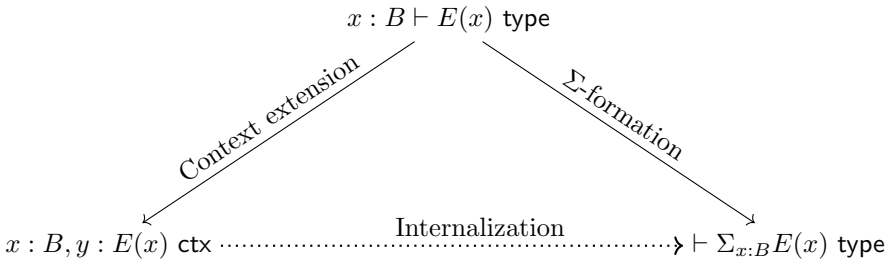

FIGURE 3. Σ-type internalizes context extension.

a term $x : a =_A b$, it suffices to assume that x is $r_a : a =_A a$. The elimination rule is also called **path induction**.

$$\frac{\vdash A\ \mathsf{type} \qquad \vdash a : A \qquad \vdash b : A}{\vdash a =_A b\ \mathsf{type}}\ \textsc{Form} \qquad\qquad \frac{\vdash a : A}{\vdash r_a : a =_A a}\ \textsc{Intros}$$

$$\frac{x : A, y : A, z : x =_A y \vdash D(x,y,z)\ \mathsf{type} \qquad x : A \vdash d(x) : D(x,x,r_x)}{x : A, y : A, z : x =_A y \vdash \mathsf{ind}_{=,d}(x,y,z) : D(x,y,z)}\ \textsc{Elim}$$

$$\frac{x : A, y : A, z : x =_A y \vdash D(x,y,z)\ \mathsf{type} \qquad x : A \vdash d(x) : D(x,x,r_x)}{x : A \vdash \mathsf{ind}_{=,d}(x,x,r_x) \doteq d(x) : D(x,x,r_x)}\ \textsc{Comp}$$

Example 3.1 (Symmetry)**.** We define a function $(-)^{-1} : (a = b) \to (b = a)$ that produces the inverse of a path. To this end, assume that $x : (a = b)$. By path induction, it suffices to assume that x is $r_a : a = a$. In this case, the goal also follows by reflexivity.

Transitivity of equality can be proved in a number of ways.

Example 3.2 (Transitivity)**.** We define a function $- \cdot - : (a = b) \to (b = c) \to (a = c)$ that produces the composite of two paths. To this end, assume $x : a = b$ and $y : b = c$. By path induction on x and y, it suffices to assume that both x and y are r_a. The result follows immediately by r_a.

We talk about judgemental equalities (e.g., $a \doteq b : A$) at a "meta" level. The identity types (e.g., $r_a : a =_A b$) internalize these into the "type-and-term" level.

The congruence rules for judgmental equalities say that

$$\frac{a : A, b : A \vdash a \doteq b : A \qquad a : A \vdash a =_A a\ \mathsf{type}}{a : A, b : A \vdash (a =_A a) \doteq (a =_A b)\ \mathsf{type}}$$

and that

$$\frac{a : A \vdash r_a : a =_A a \qquad a : A, b : A \vdash (a =_A a) \doteq (a =_A b)\ \mathsf{type}}{a : A, b : A \vdash r_a : a =_A b}$$

Thus, judgmentally equal terms are propositionally equal by reflexivity.

Exercise 3.1. Show that, for all $n : \mathbb{N}$, $\mathsf{add}\ 0\ n = n$ for the addition function defined in Example 2.2. In other words, find a term that inhabits the type

$$\prod_{n:\mathbb{N}} \mathsf{add}\ 0\ n = n$$

4. The Space Interpretation

In §3, we informally referred to terms of an identity type as paths. In this section, we explore the space interpretation of types due to the late mathematician Vladimir Voevodsky, who showed that MLTT can be interpreted in a category of spaces (the category of Kan complexes).

A type A can be interpreted as a space, and inhabitants p and q of the identity type $a =_A b$ can be interpreted as two parallel **paths** from a to b in the space A. Then since p and q are themselves terms, the equality formation rule allows us to form a new type.

MLTT	Space
types T	spaces T
terms t	points $t \in T$
dependent types $x : B \vdash E(x)$ type	fibrations $\begin{array}{c} E \\ \downarrow \\ B \end{array}$
equalities	paths/homotopies

TABLE 3. The space interpretation of MLTT.

$$\frac{\vdash p : a =_A b \qquad \vdash q : a =_A b}{\vdash p =_{a=_A b} q \text{ type}}$$

The inhabitants of the type $p =_{a=_A b} q$ can then be interpreted as **2-dimensional** paths, or **homotopies**, between the two 1-dimensional paths p and q. Similarly, we can form the type $r =_{p=_{(a=_A b)} q} s$ of 3-dimensional paths between two parallel 2-dimensional paths. This process can be repeated ad infinitum, giving higher dimensional structures. See Figure 4.

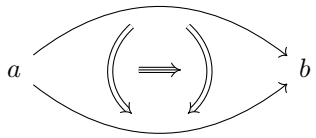

FIGURE 4. An illustration of higher dimensional paths.

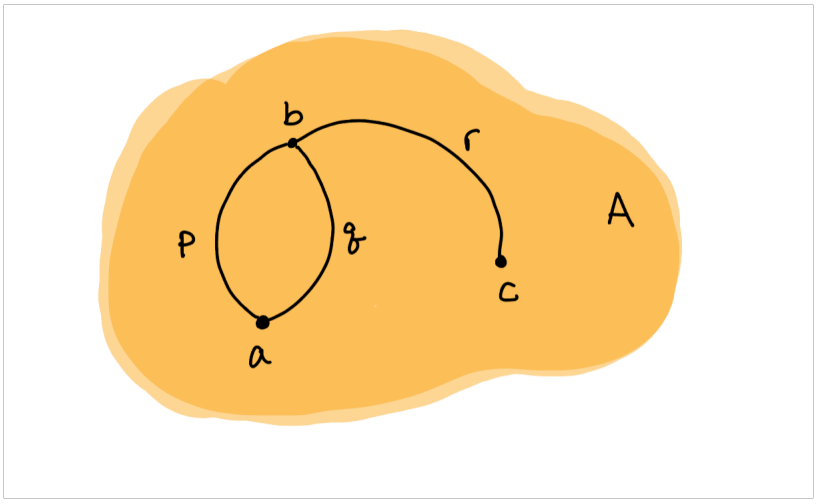

FIGURE 5. A disk-shaped space A and points a, b, and c and paths p, q, and r.

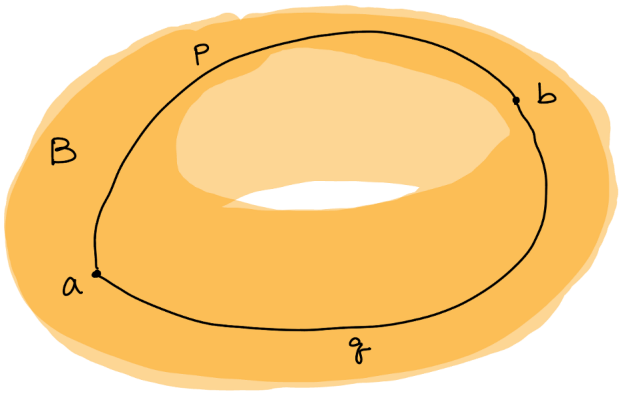

FIGURE 6. A torus-shaped space B and points a and b and paths p and q.

Reflexivity, symmetry, and transitivity of paths suggest groupoidal behaviors of types.

Equality	Homotopy
reflexivity	constant path
symmetry	inverse path
transitivity	path concatenation

TABLE 4. Examples 3.1 and 3.2 reveal the connections between identity types and homotopies.

Exercise 4.1. Show that functions $f : A \to B$ respect equalities. In other words, construct the following function:

$$\mathsf{ap}_f : (a =_A b) \to (f(a) =_B f(b))$$

The notation ap_f can be read as the action on paths of f [Uni13].

Exercise 4.2 (Transport)**.** Show that for any dependent type $x : B \vdash E(x)\ \mathsf{type}$, any terms $b, b' : B$, and any path $p : b =_B b'$, there is a function $p_* : E(b) \to E(b')$.

This ensures that every dependent type we can construct respects propositional equality. If we think of E as a predicate on B, this means that if $E(b)$ is true and $b =_B b'$, then $E(b')$ is true. This is part of a more sophisticated relationship between type theory and homotopy theory (Quillen model category theory, or QMC theory). Transport says that $\pi_1 : \Sigma_{b:B} E(b) \to B$ behaves like a fibration in a QMC. For more information, see Appendix A.

4.1. **Functional Extensionality, UIP, and Univalence.** For Bool, we can show that $\mathsf{false} = \mathsf{false}$, $\mathsf{true} = \mathsf{true}$, and that $\mathsf{false} \neq \mathsf{true}$. For $\mathbb{N}$, we can show that if

$S(n) = S(m)$, then $n = m$ for all $n, m : \mathbb{N}$, and that $0 \neq S(n)$ for all $n : \mathbb{N}$. A more involved example is the Σ-types, for which we can show that

$$(s = t) \simeq \textstyle\sum_{p:\pi_1(s)=\pi_1(t)} p_*(\pi_2(s)) = \pi_2(t) \quad \text{for all } s, t : \sum_{a:A} B(a).$$

This roughly says that the identity type $s = t$ is "equivalent" (a notion that we will introduce later) to the disjoint union of paths in the fibers.

4.1.1. *Functional Extensionality.* We consider two set-theoretic functions equal when they are pointwise equal. This property is called **functional extensionality** (FunExt). We might want Π-types to have this property, but this is **not provable** in MLTT. However, this is validated by interpretations into logic, sets, and spaces.

4.1.2. *Uniqueness of Identity Proofs.* The **Uniqueness of Identity Proofs** (UIP) states that any two proofs of an identity $p = q$ are equal. This property is **not provable** in MLTT. However, it is validated by interpretations into logic and set, but not in spaces because there may be multiple paths in a space. See Figure 6.

4.1.3. *Univalence.* From a topologist's perspective, two homeomorphic spaces are equal; from a group theorist's perspective, two isomorphic groups are equal. Similarly, two equivalent types in a universe $\mathcal{U}$ should be considered equal. **Univalence** states that

$$(T = S) \simeq (T \simeq S)$$

This is **not provable** in MLTT, but it is validated by interpretation into **spaces**. Univalence is a consequence of the **univalent axiom** (UA), which we will introduce later.

As it turns out, UA and UIP are incompatible, and both UA and UIP imply FunExt. To avoid inconsistency, we cannot have both UA and UIP. We choose UA. Both choices make sense, but choosing UIP would send us in the direction of set theory, not HoTT.

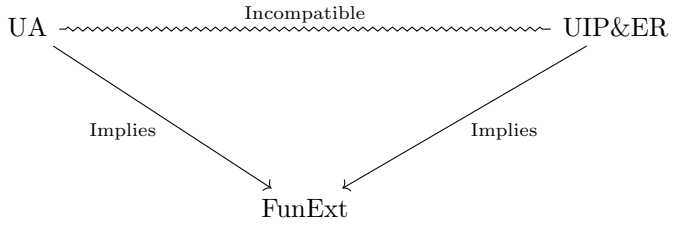

FIGURE 7. Both UA and UIP with equality reflection (ER) imply functional extensionality. However, UA and UIP are incompatible.

5. Homotopy Levels

Types in HoTT come equipped with higher-dimensional structures. In some types, however, these structures are trivial above a certain dimension. In this section, we introduce **homotopy levels**, or h-levels for short, and discuss some special cases.

5.1. **h-level 0.** A type T is **contractible** (has h-level 0) if there is a **center of contraction** t, and for every inhabitant s of that type there is a path from s to t.

$$\mathsf{isContr}(T) := \sum_{t:T} \prod_{s:T} (s = t)$$

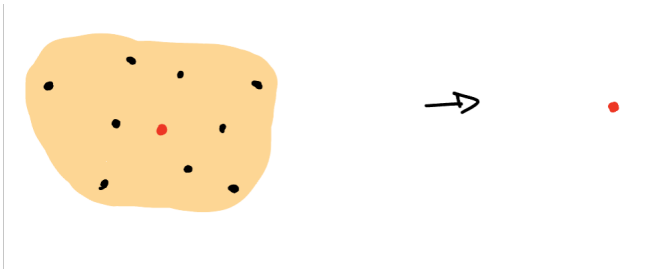

FIGURE 8. A type T with h-level 0 has a center of contraction t and every inhabitant has a path to t.

For example, the unit type $\mathbb{1}$ is contractible. Choose tt to be the center of contraction. The result follows by the elimination rule for $\mathbb{1}$.

Exercise 5.1. Show that if T is contractible and inhabited, then $T \simeq \mathbb{1}$.

5.2. **h-level 1.** A type T is a **proposition** (has h-level 1) if for any $s, t : T$ the identity type $s = t$ is contractible.

$$\mathsf{isProp}(T) := \prod_{s,t:T} \mathsf{isContr}(s = t)$$

Note that a proposition does not need to be inhabited. For example, the empty type $\varnothing$ is a proposition. See Exercise 5.2.

Exercise 5.2. Show that $\varnothing$, $\mathbb{1}$ are propositions.

Exercise 5.3. Show that any contractible type is a proposition.

Exercise 5.4. Show that if a proposition is inhabited, then it is contractible. (This says that a proposition is informally equivalent to $\varnothing$ or $\mathbb{1}$, i.e., truth values)

5.3. **h-level 2.** A type T is a **set** (has h-level 2) if for any $s, t : T$ the identity type $s = t$ is a proposition.

$$\mathsf{isSet}(T) := \prod_{s,t:T} \mathsf{isProp}(s = t)$$

The type T looks like a discrete collection of points. For example, the types $\mathbb{N}$ and Bool are sets.

5.4. **h-level 3.** A type T is a **groupoid** (has h-level 3) if for any $s, t : T$ the identity type $s = t$ is a set.

$$\mathsf{isGroupoid}(T) := \prod_{s,t:T} \mathsf{isSet}(s = t)$$

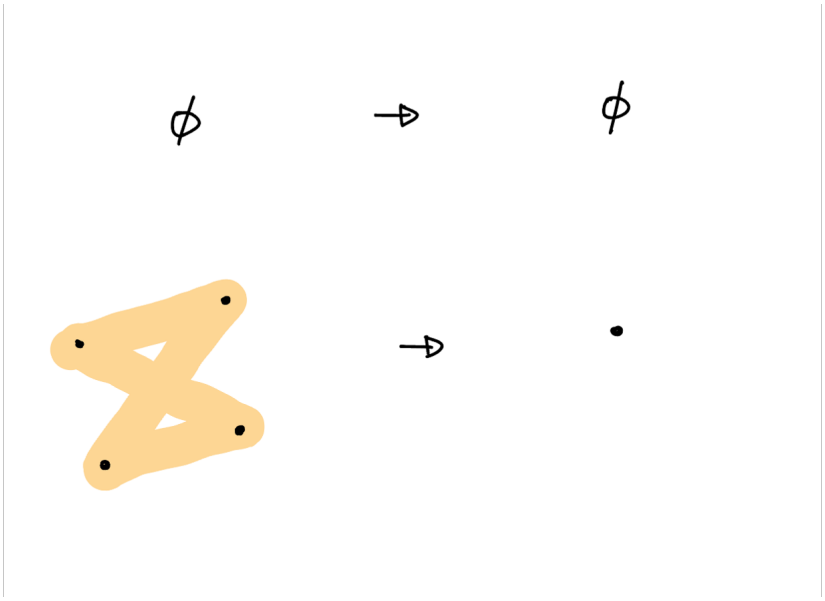

FIGURE 9. If a type T has h-level 1, then the identity type $s = t$ between any two inhabitants s and t is contractible. This means that any two proofs of a proposition are equal.

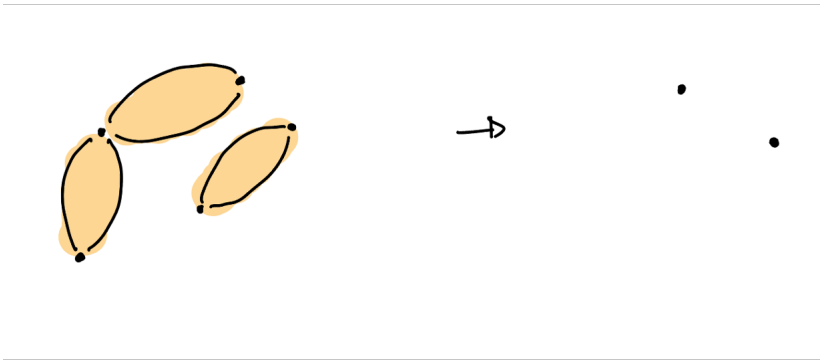

FIGURE 10. If a type T has h-level 2, then the identity type $s = t$ between any two inhabitants s and t is a proposition. Thus, T looks like a set.

In general, $\mathsf{isofhlevel}\ n\ T$ is defined recursively as follows:

$$\mathsf{isofhlevel}\ 0\ T := \mathsf{isContr}(T)$$
$$\mathsf{isofhlevel}\ S(n)\ T := \prod_{s,t:T} \mathsf{isofhlevel}\ n\ (s = t)$$

Exercise 5.5. Show that if a type T has h-level n, then it has h-level $n + 1$.

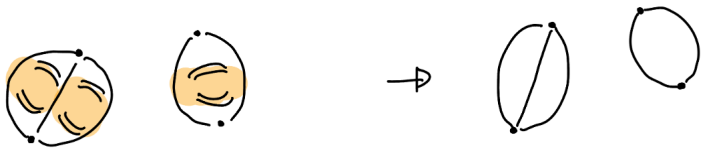

FIGURE 11. If a type T has h-level 3, then the identity type $s = t$ between any two inhabitants s and t is a set. Thus, T looks like a groupoid.

6. Equivalences

Given a function $f : A \to B$, we can define $\mathsf{isEquiv}(f)$ by

$$\sum_{g:B\to A} (g \circ f = 1_A) \times (f \circ g = 1_B)$$

Sadly, this type is not a proposition. Thus, it has nontrivial high-dimensional structures that we don't want. To remedy this, recall that a set-theoretic function $f : A \to B$ is a bijection if for all $b \in B$, the fiber over b is a singleton set. This suggests an alternative definition for $\mathsf{isEquiv}(f)$.

Definition 6.1.

$$\mathsf{isEquiv}(f) := \prod_{b:B} \mathsf{isContr}\left(\sum_{a:A} f(a) = b\right)$$

This states that for every $b : B$, the fiber over b is contractible, i.e., there exists a unique element in the preimage of f. One can prove that $\mathsf{isEquiv}(f)$ is a proposition.

Definition 6.2 (Equivalence)**.** Given any two types $A, B : \mathcal{U}$, we write $A \simeq B$ to denote the type $\sum_{f:A\to B} \mathsf{isEquiv}(f)$.

For any two types $A, B : \mathcal{U}$, we can construct a function $\mathsf{idToEquiv} : (A = B) \to (A \simeq B)$ by path induction.

Definition 6.3 (Univalence Axiom)**.** The **univalence axiom** asserts that

$$\mathsf{ua} : \mathsf{isEquiv}(\mathsf{idToEquiv})$$

An immediate consequence of this axiom is that $(A = B) \simeq (A \simeq B)$.

6.1. Univalence for Logic and Sets.

Definition 6.4. We define the type of propositions as the following sigma type:

$$\mathsf{Prop} := \sum_{P:\mathcal{U}} \mathsf{isProp}(P)$$

The univalence axiom implies that the type Prop is univalent. That is, for any $P, Q : \mathsf{Prop}$,

$$(P = Q) \simeq (P \leftrightarrow Q)$$

As a corollary, the type Prop is a set rather than a proposition because there are propositions that are not interprovable, but all the interprovable ones can be identified. We define the type of sets in the same fashion.

Definition 6.5.

$$\mathsf{Set} := \sum_{S:\mathcal{U}} \mathsf{isSet}(S)$$

Similarly, the univalence axiom implies that Set is univalent.

$$(S = T) \simeq (S \cong T)$$

As a consequence, Set is a groupoid because, for example, the set containing two elements is isomorphic to itself in two different ways.

With Set in hand, we can define the type of groups. Recall that a group consists of a carrier set G, a neutral element e, a multiplication operator $\cdot$, and an inverse operator $-^{-1}$ such that

(1) $e \cdot g = g$ for all $g \in G$,
(2) $g \cdot e = g$ for all $g \in G$,
(3) $(g \cdot h) \cdot k = g \cdot (h \cdot k)$ for all $g, h, k \in G$,
(4) $g^{-1} \cdot g = e$ for all $g \in G$, and
(5) $g \cdot g^{-1} = e$ for all $g \in G$.

Definition 6.6.

$$\begin{aligned}\mathsf{Grp} := \sum_{G:\mathsf{Set}} \sum_{e:G} \sum_{\cdot:G\to G\to G} \sum_{-^{-1}:G\to G} &\left(\prod_{g:G} e \cdot g = g\right) \times \\ &\left(\prod_{g:G} g \cdot e = g\right) \times \left(\prod_{g,h,k:G} (g \cdot h) \cdot k = g \cdot (h \cdot k)\right) \times \\ &\left(\prod_{g:G} g^{-1} \cdot g = e\right) \times \left(\prod_{g:G} g \cdot g^{-1} = e\right)\end{aligned}$$

The role of the inhabitants of the group axioms is to serve as a **witness** that the neutral element, the multiplication, and the inverse of a group interact properly. We don't want the group axioms to carry additional structures. Thus, the carrier G needs to be a set to ensure that the group axioms are propositions.

Again, the univalence axiom implies that for any two groups, G and H,

$$(G = H) \simeq (G \cong H).$$

Similarly, we can define the type of groupoids by

$$\mathsf{Grpd} := \sum_{T:\mathcal{U}} \mathsf{isGroupoid}(T)$$

And the type Grpd is univalent. In fact, any algebraic structure on a set has a univalence result [CD13].

The moral of the story is that univalence allows us to do mathematics up to some appropriate notion of "sameness". For example, the appropriate notion of sameness for groups is group isomorphisms. The **Structure Identity Principle**, or **SIP** (Aczel, Coqand), is the idea that an isomorphism (or our notions of equivalence) should respect the structure of a given type. The **Identity of Indiscernables**

(Leibniz) states that any two objects that have all their properties in common cannot be distinct. Univalence allows us to treat equivalent types as exactly equal, incorporating these principles into our theory.

7. Higher Inductive Types

An ordinary inductive type, such as Bool, is freely generated by its canonical terms true and false. Since we now consider types as having terms, equalities, equalities between equalities, etc, we can consider **higher inductive types** whose constructors can generate terms, equalities, equalities between equalities, etc. For example, we can define the type D^1 to be the higher inductive type generated by two points $\bullet$ and $\star$, and a path between them $p : \bullet = \star$. The topological intuition here is that we connect the points $\bullet$ and $\star$ by adding the path p. (S^0 is the 0-dimensional sphere, and D^1 is the the 1-dimensional disc.)

$$\frac{}{\vdash D^1\ \mathsf{type}}\ \textsc{Form} \qquad \frac{}{\vdash \bullet : D^1 \qquad \vdash \star : D^1 \qquad \vdash p : \bullet = \star}\ \textsc{Intros}$$

$$\frac{x : D^1 \vdash E(x)\ \mathsf{type} \qquad \vdash t : E(\bullet) \qquad \vdash f : E(\star) \qquad \vdash \pi : p_* t = f}{x : D^1 \vdash \mathsf{ind}_{D^1,t,f,\pi}(x) : E(x)}\ \textsc{Elim}$$

$$\frac{x : D^1 \vdash E(x)\ \mathsf{type} \qquad \vdash t : E(\bullet) \qquad \vdash f : E(\star) \qquad \vdash \pi : p_* t = f}{\begin{array}{c}\vdash \mathsf{ind}_{D^1,t,f,\pi}(\bullet) \doteq t : E(\bullet) \qquad \vdash \mathsf{ind}_{D^1,t,f,\pi}(\star) \doteq f : E(\star) \\ \vdash \mathsf{ind}_{D^1,t,f,\pi}(p) = \pi : p_* t = f\end{array}}\ \textsc{Comp}$$

We explore a few more higher inductive types.

7.1. Propositional Truncation.

Given two propositions P and Q, we can prove that the types $P \times Q$, $P \to Q$, and $\neg P$ are propositions. Thus, we can define conjunctions, implications, and negations in the most obvious way. However, $P + Q$ is not necessarily a proposition since if we have $p : P$ and $q : Q$, then we have $\mathsf{inl}(p) : P + Q$ and $\mathsf{inr}(q) : P + Q$. We can turn this into a proposition by identifying $\mathsf{inl}(p)$ and $\mathsf{inr}(q)$.

Definition 7.1. In general, given a type T, the **propositional truncation** $\|T\|_1$ of T is the higher inductive type generated by an injection $|\cdot|_1 : T \to \|T\|_1$ and a path identifying any pair of inhabitants:

$$p : \prod_{x,y:T} |x|_1 = |y|_1$$

With this, we can define $P \vee Q := \|P + Q\|_1$.

Exercise 7.1. Show that for all T, $\mathsf{isProp}(\|T\|_1)$.

7.2. Set Truncation.

We can define the type of circles S^1 to be the higher inductive type generated by a base point $b : S^1$ and a loop $\ell : b = b$. A classical result in algebraic topology is that the fundamental group of the circle is isomorphic to $\mathbb{Z}$. Indeed, we have that

$$\pi_1(S^1) \simeq \mathbb{Z}$$

We define $\pi_1(T, t) := (S^1, b) \to (T, t)$ to be the "set" of base-perserving functions. However, there no reason for this type to be a set. Set truncation allows us to remedy this.

Definition 7.2. Given a type T, the **set truncation** $\|T\|_2$ of T is the higher inductive type generated by an injection $|\cdot|_2 : T \to \|T\|_2$, and a 2-path identifying any two 1-paths:

$$p : \prod_{x,y:T} \prod_{p,q:x=y} |p|_2 = |q|_2$$

Now, we can define $\pi_1(T,t) := \|(S^1, b) \to (T,t)\|_2$.

Exercise 7.2. Show that for all T, $\mathsf{isSet}(\|T\|_2)$.

7.3. **Quotient Types.**

Definition 7.3. Given an equivalence relation $\sim$ on a set S, we can take the **quotient** $S/\sim$ to be the higher inductive type given by:

- an injection $\iota : S \to S/\sim$;
- a dependent function $j : \prod_{x,y:S} x \sim y \to \iota(x) = \iota(y)$;
- constructors for $\|\cdot\|_2$.

8. Category Theory

Traditionally, a category $\mathbf{C}$ consists of

(1) a set $\mathrm{ob}(\mathbf{C})$ of objects,
(2) a set $\hom(X,Y)$ of morphisms for all $X, Y \in \mathrm{ob}(\mathbf{C})$,
(3) an identity morphism $\mathbf{1}_X \in \hom(X,X)$ for all $X \in \mathrm{ob}(\mathbf{C})$,
(4) a composition function $\circ : \hom(Y,Z) \to \hom(X,Y) \to \hom(X,Z)$ for all $X, Y, Z \in \mathrm{ob}(\mathbf{C})$.

These data are subject to the identity and associativity laws.

We could talk about categories in the set level (like most classical mathematics). However, the structural identity principle for structures on sets tells us that

$$(\mathbf{C} =_{\mathsf{Cat}} \mathbf{D}) \simeq (\mathbf{C} \cong \mathbf{D}).$$

But isomorphism is not the right kind of sameness for categories; they are too strong. Instead of squashing everything down to the level of sets, HoTT affords higher dimensional structures.

8.1. **Univalent Categories.** Observe that every category has a "core groupoid" contained within it: the objects and all isomorphisms. With a groupoid of objects in hand, we can then glue additional morphisms to this groupoid, forming a category.

Definition 8.1. A **univalent category** $\mathbf{C}$ consists of:

- a groupoid of objects $\mathrm{ob}(\mathbf{C}) : \mathsf{Grpd}$;
- a set $\hom(X,Y)$ for every pair $X, Y : \mathrm{ob}(\mathbf{C})$, i.e.,

$$X, Y : \mathrm{ob}(\mathbf{C}) \vdash \hom(X,Y) : \mathsf{Set};$$

- a term $\mathbf{1}_X : \hom(X,X)$ for every $X : \mathrm{ob}(\mathbf{C})$, i.e.,

$$X : \mathrm{ob}(\mathbf{C}) \vdash \mathbf{1}_X : \hom(X,X);$$

- a function $\circ : \hom(Y,Z) \to \hom(X,Y) \to \hom(X,Z)$ for every $X, Y, Z : \mathrm{ob}(\mathbf{C})$, i.e.,

$$X, Y, Z : \mathrm{ob}(\mathbf{C}) \vdash \circ : \hom(X,Y) \to \hom(Y,Z) \to \hom(X,Z)$$

These data are subject to the usual axioms of categories and that the function

$$\mathsf{idToIso} : (X = Y) \to \mathsf{Iso}(X, Y)$$

is an equivalence, where

$$\mathsf{Iso}(X, Y) := \sum_{f:\hom(X,Y)} \sum_{g:\hom(Y,X)} (g \circ f = \mathbf{1}_X) \times (f \circ g = \mathbf{1}_Y).$$

Thus, the right notion of sameness for objects is isomorphism. This additional requirement is often called (internal) univalence.

The type of categories Cat can then be defined as an iterated sigma type. As a consequence of the univalence axiom,

$$(\mathbf{C} =_{\mathsf{Cat}} \mathbf{D}) \simeq (\mathbf{C} \simeq \mathbf{D}).$$

Thus, the terms of Cat are categories and the equalities are equivalence of categories.

Corollary. Cat is a 2-groupoid (h-level 4).

This is a great achievement for the univalent foundations. Category theorists call definitions that are not invariant under equivalence of categories **evil**. This definition helps us avoid "evilness" when we talk about categories.

8.2. **Rezk Completion.** A **precategory** is a univalent category except that the type of objects $\mathrm{ob}(\mathbf{C})$ is not necessarily a groupoid, and that internal univalence is not assumed. There is a canonical way of turning a precategory into a univalent one.

Definition 8.2. The **Rezk completion** of a precategory $\mathbf{C}$ takes $\mathrm{ob}(\mathsf{Rezk}(\mathbf{C}))$ to be the higher inductive type given by:

- an injection $\iota : \mathrm{ob}(\mathbf{C}) \to \mathrm{ob}(\mathsf{Rezk}(\mathbf{C}))$;
- a dependent function $j : \prod_{X,Y:\mathrm{ob}(\mathbf{C})} \mathsf{Iso}(X, Y) \to \iota(X) = \iota(Y)$;
- constructors for $\| \cdot \|_3$

References

[CD13] Thierry Coquand and Nils Anders Danielsson. Isomorphism is equality. *Indagationes Mathematicae*, 24(4):1105–1120, 2013. In memory of N.G. (Dick) de Bruijn (1918–2012).

[ML96] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

[Uni13] Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `http://homotopytypetheory.org/book/`, first edition, 2013.

Appendix A. Transport and Fibrations

As mentioned earlier, the ability to transport along an equality corresponds to the projection map $\pi_1 : \Sigma_{b:B} E(b)$ behaving like a fibration in a QMC. We will now discuss one way to see this, taking the liberty to interpret equalities as a map from the interval type into the target type, as is possible in a QMC (as well as in cubical formulations of HoTT).

Consider Figure 12. If the outer diagram commutes, we have the data $b : B$, $e : E(b)$, and a path $p : b = b'$ for some (unstated) $b' : B$.

This follows because, when interpreting p as a map from I to B, evaluating p at $0 : I$ must result in b by commutativity.

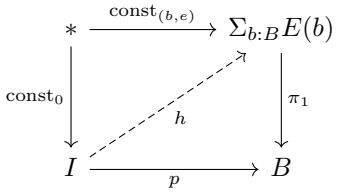

FIGURE 12. Given that the outer diagram commutes, we have that if π_1 is a fibration, there exists a homotopy lifting map h making everything commute.

Then, if π_1 is a fibration, we get that there exists a homotopy lift $h : (b, e) = (b', e')$ for some (unstated) $e' : E(b')$ that makes the diagram commute.

This is because, when interpreting h as a map from I to $\Sigma_{b:B}E(b)$, evaluating p at $0 : I$ must result in (b, e) by commutativity.

Furthermore, evaluating h at $1 : I$ and then applying π_1 must result in the same value as evaluating p at $1 : I$, also by commutativity.

Thus, given such an h, we can define $tr_p(e)$ to equal $\pi_2(h(1)) : E(b')$. Thus, if π_1 is always a fibration, given a term $p : b = b'$, we can define $tr_p : E(b) \to E(b')$.

In the reverse direction, given a term $tr_p : E(b) \to E(b')$ along with the property that $tr_{r_b} \doteq \text{id}_{E(b)}$ (which holds when defining transport using path induction), given that the outer diagram in Figure 12 commutes, we can define $h : (b, e) = (b', tr_p(e))$.

To do this, we can use the elimination rule for identity types to reduce this problem to the case where $b' \doteq b$ and $p \doteq r_b$. As such, we now want to define $h : (b, e) = (b', tr_{r_b}(e))$.

But as $(b', tr_{r_b}(e))$ definitionally reduces to $(b, \text{id}_{E(b)}(e))$ and then again to (b, e), we get that $h : (b, e) = (b, e)$.

As such, we can just let h be $r_{(b,e)}$ to finish the proof.

Thus, we have that if π_1 is a fibration, we can construct transport along a path $p : b = b'$ as $tr_b : E(b) \to E(b')$, and if we have transport we can show that π_1 is a fibration.

As a bonus exercise, prove that this map taking in tr_p and output h is an equivalence!

APPENDIX B. DIGRESSION: PI-TYPES

$\prod$-types are dependent function types. Note that they are **not** an inductive type, but we include a sketch of the rules for them here.

$$\frac{x : B \vdash E(x)\ \mathsf{type}}{\vdash \prod_{x:B} E(x)\ \mathsf{type}}\ \textsc{Form} \qquad \frac{x : B \vdash e : E(x)}{\vdash \lambda x : B.\ e : \prod_{x:B} E(x)}\ \textsc{Intros}$$

$$\frac{\vdash f : \prod_{x:B} E(x) \qquad \vdash b : B}{\vdash f\ b : E(b)}\ \textsc{Elim} \qquad \frac{x : B \vdash e : E(x) \qquad \vdash b : B}{\vdash (\lambda x : B.\ e)\ b \doteq e[b/x] : E(b)}\ \textsc{Comp}$$

$$\frac{\vdash f : \prod_{x:B} E(x)}{\vdash f \doteq (\lambda x : B.\ f\ x) : \prod_{x:B} E(x)}\ \textsc{Uniq}$$

Appendix C. Links and FAQ

(1) **How does univalence imply functional extensionality?** Here is a blog post detailing that: Another proof that univalence implies functional extensionality by Dan Licata.

Perhaps the biggest takeaway from this article is that this proof that univalence implies functional extensionality is very closely related to the proof in cubical type theory of functional extensionality, which doesn't use univalence! There is something bigger at play here, demonstrating that univalence in some way makes things in MLTT "work better" than it used to.

Other than that, I highly recommend checking out the article linked above, as it is by far the clearest explanation of this proof that I've seen.

(2) **Are h-levels different from universe levels?** Yes! In a type hierarchy such as the one used by Coq, each $\texttt{Type}_i$ has h-level ∞.

(3) **How does the inductive definition of identity types allow for non-canonical terms?** The crucial difference for identity types is that we are inductively defining a *family* of types in **??**. The types build by these definitions are indexed by the terms being equated, allowing us to have non-canonical terms.