

## Abstract

This is a brief, introductory overview of homotopy type theory (HoTT). After covering some preliminary concepts from type theory, we work to state the *univalence axiom*, a powerful feature of HoTT's syntax as formulated by Voevodsky. Along the way, we develop the notion of an identity type and of type equivalence by way of homotopy theory (an area of algebraic topology). Afterward, we mention some categorical models of HoTT.

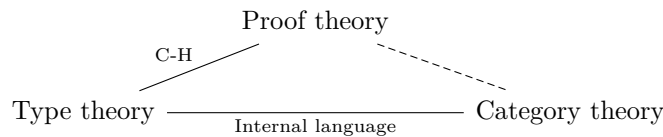
## Contents

|                                     |           |
|-------------------------------------|-----------|
| <b>Introduction</b>                 | <b>2</b>  |
| <b>1 Elements of type theory</b>    | <b>2</b>  |
| <b>2 Identity types</b>             | <b>4</b>  |
| <b>3 Type-theoretic equivalence</b> | <b>8</b>  |
| <b>4 Univalence</b>                 | <b>10</b> |
| <b>5 Models of type theory</b>      | <b>12</b> |
| <b>Further topics</b>               | <b>12</b> |
| <b>Sources</b>                      | <b>13</b> |

## Introduction

In its current form, HoTT largely originated from Vladimir Voevodsky, a Fields Medalist who passed away in 2017. He wanted a logical system that could both handle certain higher-dimensional algebraic objects and facilitate formalization better than set theory could. In 2005 and 2006, the first homotopical models of intensional (dependent) type theory were constructed by Steve Awodey, Michael Warren, and Voevodsky. In 2012-13, the IAS at Princeton held *A Special Year on Univalent Foundations of Mathematics*, from which the standard reference *Homotopy Type Theory: Univalent Foundations of Mathematics* was produced. This includes Voevodsky's univalence axiom in its formulation of type theory.

One may think of HoTT as a homotopical version of what Robert Harper calls *computational trinitarianism*:



## 1 Elements of type theory

We are given a formal language  $\mathcal{L}$  consisting of certain terms, say, a variant of the untyped lambda calculus. We can enrich  $\mathcal{L}$  with additional primitive objects called *types*. If  $a$  is a term and  $A$  a type, then we write  $a : A$  to express the *judgment* (distinct from a proposition) that  $a$  *inhabits* / *has type*  $A$ . We declare that types themselves inhabit types known as *universes*, which are arranged in a cumulative hierarchy.

$$\mathbf{Type}_0 : \mathbf{Type}_1 : \mathbf{Type}_2 : \cdots$$

If instead we declared a single universe **Type** that every type inhabits, then we could encode Russel's paradox into our type theory. Our hierarchy avoids such a problem by tracking which level a newly formed type inhabits. For convenience, however, we usually avoid writing this level explicitly and use the term **Type** instead. In this sense, **Type** is known as a *universe a la Russell*.

**Example 1.0.1.** The following (among others) will be base types of our language.

1. The *empty type*  $\perp$ .
2. The *singleton type* **unit**.

We write  $a \equiv b : A$  to express that  $a$  and  $b$  are *definitionally equal terms type*  $A$ . We declare that  $\equiv$  is an equivalence relation with respect to any type  $A$ . We also declare that if both  $A \equiv B : \mathbf{Type}$  and  $a : A$  are derivable judgments, then so is  $a : B$ .

**Definition 1.0.2.** Let  $A$  be any type and  $B : A \rightarrow \mathbf{Type}$  be any family of types (e.g.,  $\lambda n. T^n : \mathbb{N} \rightarrow \mathbf{Type}$ ).

1. We form the *dependent product type*  $\prod_{x:A} B(x)$  according to the following four inference rules.

- (a)  $\prod$ -Introduction: If  $b : B(x)$  for any  $x : A$ , then  $\lambda x. b : \prod_{x:A} B(x)$ .
- (b)  $\prod$ -Elimination: If  $f : \prod_{x:A} B(x)$  and  $a : A$ , then  $f(a) : B(a)$ .

- (c)  $\prod$ -Computation: If  $b : B(x)$  for any  $x : A$  and  $a : A$ , then  $(\lambda x.b)(a) \equiv b[x := a] : B(a)$ .
- (d)  $\prod$ -Uniqueness: If  $f : \prod_{x:A} B(x)$ , then  $f \equiv \lambda x.f(x) : \prod_{x:A} B(x)$ .

Note that the ordinary function type  $A \rightarrow B$  is a special case of the dependent product type. Intuitively, a dependent product type is like a set-theoretic choice function.

2. Now, add the term  $\text{ind}_{\sum_{x:A} B(x)}(t_1, t_2, t_3)$  to  $\mathcal{L}$  where  $t_1, t_2$ , and  $t_3$  are any given terms of  $\mathcal{L}$ .

We form the *dependent sum type*  $\sum_{x:A} B(x)$  according to the following three inference rules.

- (a)  $\sum$ -Introduction: If  $a : A$  and  $b : B(a)$ , then  $(a, b) : \sum_{x:A} B(x)$ .
- (b)  $\sum$ -Elimination: Given any family of types  $C : (\sum_{x:A} B(x)) \rightarrow \mathbf{Type}$ , if  $g : C(x, y)$  for any  $x : A$  and  $y : B(x)$  and  $p : \sum_{x:A} B(x)$ , then  $\text{ind}_{\sum_{x:A} B(x)}(C, g, p) : C(p)$ .
- (c)  $\sum$ -Computation: Given any family of types  $C : (\sum_{x:A} B(x)) \rightarrow \mathbf{Type}$ , if  $g : C(x, y)$  for any  $x : A$  and  $y : B(x)$ ,  $a : A$ , and  $b : B(a)$ , then  $\text{ind}_{\sum_{x:A} B(x)}(C, g, (a, b)) \equiv g(a, b) : C(a, b)$ .

Note that the ordinary product type  $A \times B$  is a special case of the dependent sum type. Intuitively, a dependent sum type is like a set-theoretic disjoint union (or, more generally, a coproduct).

**Definition 1.0.3.**

1. Define the *left projection function*  $\text{pr}_1$  by the judgments

$$\text{pr}_1 : \left( \sum_{x:A} B(x) \right) \rightarrow A \quad \text{pr}_1(a, b) \equiv a : A.$$

2. Define the *right projection function*  $\text{pr}_2$  by the judgements

$$\text{pr}_2 : \prod_{p : \sum_{x:A} B(x)} B(\text{pr}_1(p)) \quad \text{pr}_2(a, b) \equiv b : B(a).$$

Given  $p : \sum_{x:A} B(x)$ , we have that  $p \equiv (\text{pr}_1(p), \text{pr}_2(p)) : \sum_{x:A} B(x)$ . This is known as the uniqueness principle for dependent sum types.

**Note 1.0.4 (Curry-Howard isomorphism).** It can be useful to think of our type theory as familiar first-order logic.

$$\text{False} \longleftrightarrow \perp$$

$$\text{True} \longleftrightarrow \text{unit}$$

$$P \wedge Q \longleftrightarrow P \times Q$$

$$P \vee Q \longleftrightarrow P + Q \quad .$$

$$P \implies Q \longleftrightarrow P \rightarrow Q$$

$$\forall x. P(x) \longleftrightarrow \prod_{x:A} P(x)$$

$$\exists x. P(x) \longleftrightarrow \sum_{x:A} P(x)$$

## 2 Identity types

**Definition 2.0.1 (Martin-Löf).** Let  $A$  be a type. The *identity type*  $\text{Id}_A : A \rightarrow A \rightarrow \mathbf{Type}$  is given inductively by the single constructor

$$\text{refl} : \prod_{a:A} \text{Id}_A(a, a).$$

We interpret  $A$  to be a topological space and each inhabitant of  $\text{Id}_A(x, y)$  to be a path from the point  $x$  to the point  $y$ . The term  $\text{refl}_a$  is thought of as the constant path at the point  $a$ . With this in mind, let  $x \rightsquigarrow_A y$  denote the type  $\text{Id}_A(x, y)$ .

The type  $x \rightsquigarrow_A y$  can be thought of as the path space of  $A$ , which consists of the set of paths in  $A$  equipped with the compact-open topology.

Under the Curry-Howard isomorphism, each path from  $x$  to  $y$  corresponds to a proof of the proposition that  $x$  equals  $y$ . HoTT is an *intensional* (as opposed to extensional) type theory in that it both distinguishes between definitional and propositional equality and allows an identity type to be inhabited by more than one term.

**Note 2.0.2 (Path induction).** Given a dependent type  $D : \prod_{\{x,y:A\}} (\text{Id}_A(x, y) \rightarrow \mathbf{Type})$ , suppose that we have constructed a term  $d : \prod_{a:A} D(a, a, \text{refl}_a)$ . Then the induction principle for identity types states that we can construct a *section*

$$J(D, d) : \prod_{\substack{\{x,y:A\} \\ p:\text{Id}_A(x,y)}} D(x, y, p)$$

of  $D$  such that  $J(D, d, \text{refl}_a) \equiv d(a)$  for each  $a : A$ .

**Lemma 2.0.3.** *Let  $A : \mathbf{Type}$  and  $x, y : A$ . Then there is some function  $\text{sym} : (x \rightsquigarrow y) \rightarrow (y \rightsquigarrow x)$  such that  $\text{sym}(\text{refl}_x) \equiv \text{refl}_x$  for each  $x : A$ . Let  $p^{-1} := \text{sym}(p)$ .*

*Proof.* Define  $D : \prod_{x,y:A} (x \rightsquigarrow y) \rightarrow \mathbf{Type}$  by the judgment

$$D(x, y, p) \equiv y \rightsquigarrow x.$$

By our construction of  $\text{Id}_A$ , we have the function

$$d := \lambda x. \text{refl}_x : \prod_{x:A} D(x, x, \text{refl}_x).$$

By induction, we obtain an extension  $J$  of  $d$  such that  $J(D, d, x, y, p) : y \rightsquigarrow x$  for every  $p : x \rightsquigarrow y$ . Choosing  $J(D, d, x, y, p)$  to be  $\text{inv}(p)$  completes our proof.  $\square$

**Lemma 2.0.4.** *Let  $A : \mathbf{Type}$  and  $x, y, z : A$ . Then there is some function*

$$\text{trans} : (x \rightsquigarrow y) \rightarrow (y \rightsquigarrow z) \rightarrow (x \rightsquigarrow z)$$

*such that  $\text{trans}(q, \text{refl}_z) \equiv q$  for any  $q : y \rightsquigarrow z$ . Let  $p * q := \text{trans}(p, q)$ .*

*Proof.* Define the dependent type  $D$  over  $\text{Id}_A$  by

$$D(x, y, q) \equiv (y \rightsquigarrow z) \rightarrow (x \rightsquigarrow z).$$

We have the function

$$d := \lambda z. \text{idmap}_{y \rightsquigarrow z} : \prod_{z:A} D(z, z, \text{refl}_z)$$

where  $\text{idmap}_{y \rightsquigarrow z} := \lambda q. q$ . By induction, we get a suitable section  $J$  of  $D$  such that

$$J(D, d, p) : (y \rightsquigarrow z) \rightarrow (x \rightsquigarrow z)$$

for each  $p : x \rightsquigarrow y$ . Finally, let  $\text{trans}(p, -) := (D, d, p)(-)$ .  $\square$

Furthermore, we can use the induction principle to show that  $A$  is a groupoid whose structure is exactly similar to the groupoid structure of a topological space  $X$ , which we recall now.

- Let  $\gamma : I \rightarrow X$  in  $X$  be a path in  $X$ . Then  $\bar{\gamma}(t) := \gamma(1 - t)$  is a path from  $\gamma(1)$  to  $\gamma(0)$ , called the *inverse* of  $\gamma$ . This corresponds to the function  $\text{sym}$ .
- Let  $\phi, \psi : I \rightarrow X$  be paths in the topological space  $X$  such that  $\phi(1) = \psi(0)$ . Then  $\phi * \psi : I \rightarrow X$  defined by

$$t \mapsto \begin{cases} \phi(2t) & 0 \leq t \leq \frac{1}{2} \\ \psi(2t - 1) & \frac{1}{2} \leq t \leq 1 \end{cases}$$

is a path from  $\phi(0)$  to  $\psi(1)$ . This corresponds to the function  $\text{trans}$ .

- Define the *fundamental groupoid of a space  $X$*  as the category  $\Pi_1(X)$  with  $\text{ob}(\Pi_1(X)) \equiv X$  and  $\text{Hom}_{\Pi_1(X)}(x, y) \equiv \{[\gamma]_{\simeq_p} \mid \gamma \text{ is a path from } x \text{ to } y\}$ . The composition of morphisms is given by the operation  $*$ .

It is straightforward but tedious to verify that  $\Pi_1$  satisfies the definition of a groupoid with  $[\bar{\gamma}] = [\gamma]^{-1}$ .

*Remark 2.0.5.* It is known that the semantic structure of Martin-Löf identity types is that of  $\infty$ -groupoids (i.e.,  $\infty$ -categories in which any  $k$ -morphism is an equivalence).

Our next result shows that if two terms of type  $A$  are propositionally equal and  $P$  is a property of inhabitants of  $A$ , then  $P$  is true of one if and only if it is true of the other.

**Lemma 2.0.6 (Transport).** *Let  $P$  be a dependent type over  $A$ . Suppose that  $p : x \rightsquigarrow_A y$ . Then there exists a function  $\text{transport}(p) : P(x) \rightarrow P(y)$  such that  $\text{transport}(\text{refl}_x)(u) \equiv u$  for any  $u : P(x)$ .*

*Proof.* Define  $D : \prod_{x,y:A} ((x \rightsquigarrow y) \rightarrow \mathbf{Type})$  by  $D(x, y, p) \equiv (P(x) \rightarrow P(y))$ . We have the function  $d := \lambda x. \text{idmap}_{P(x)} : \prod_{x:A} D(x, x, \text{refl}_x)$ . By induction, we get a section  $J$  of  $D$  that extends  $d$  and produces terms  $J(D, d, p) : P(x) \rightarrow P(y)$  for each  $p : x \rightsquigarrow y$ . Finally, take the term  $J(D, d, p)$  for  $\text{transport}(p)$ .  $\square$

*Notation.* Let  $p \cdot u := \text{transport}(p)(u)$ .

**Lemma 2.0.7.** *Let  $f : \prod_{x:A} P(x)$  and  $p : x \rightsquigarrow_A y$ . Then there is some path  $f(p) : p \cdot f(x) \rightsquigarrow_{P(y)} f(y)$ .*

*Proof.* Define  $D : \prod_{x,y:A} ((x \rightsquigarrow y) \rightarrow \mathbf{Type})$  by  $D(x, y, p) \equiv p \cdot f(x) \rightsquigarrow f(y)$ . Then

$$D(x, y, \text{refl}_x) \equiv \text{refl}_x \cdot f(x) \rightsquigarrow f(x) \equiv f(x) \rightsquigarrow f(x).$$

We thus obtain the term

$$d := \lambda x. \text{refl}_{f(x)} : \prod_{x:A} D(x, x, \text{refl}_x).$$

Applying induction finishes our proof.  $\square$

**Corollary 2.0.8.** *Any non-dependent function  $f : A \rightarrow B$  is functorial, i.e., preserves paths.*

*Proof.* Simply recall that any non-dependent function is a special case of a dependent one.  $\square$

This means that  $f$  is continuous in a certain sense.

**Lemma 2.0.9 (Path lifting).** *Let  $P$  be a dependent type over  $A$ . Suppose that  $p : x \rightsquigarrow_A y$  and that  $u : P(x)$ . Then there is some path*

$$p_\Sigma(u) : (x, u) \rightsquigarrow_{\sum_{x:A} P(x)} (y, p \cdot u).$$

*Proof.* Define  $D : \prod_{x,y:A} ((x \rightsquigarrow y) \rightarrow \mathbf{Type})$  by

$$D(x, y, p) \equiv \prod_{u:P(x)} (x, u) \rightsquigarrow (y, p \cdot u).$$

Since  $\text{refl}_x \cdot u \equiv u$  for any  $u : P(x)$ , we have that  $D(x, x, \text{refl}_x) \equiv \prod_{u:P(x)} (x, u) \rightsquigarrow (x, u)$ . But then

$$d := \lambda x \lambda u. \text{refl}_{(x,u)} : \prod_{x:A} D(x, x, \text{refl}_x).$$

By induction, we obtain some path  $p_\Sigma(u) : (x, u) \rightsquigarrow (y, p \cdot u)$  for each  $p : x \rightsquigarrow y$ .  $\square$

As fiber bundles from topology possess the homotopy lifting property, Lemma 2.0.7 encourages us to interpret the dependent sum  $\sum_{x:A} P(x)$  as a fiber bundle over  $A$ .

Let  $f, g : \prod_{x:A} P(x)$  where  $P$  is a dependent type over  $A$ . By an easy use of path induction, any term  $\alpha : f \rightsquigarrow g$  induces paths  $\alpha(x) : f(x) \rightsquigarrow_{P(x)} g(x)$  for each  $x : A$ . This yields a function

$$\text{hApply} : \prod \{A : \mathbf{Type}\} \{P : A \rightarrow \mathbf{Type}\} \left( f, g : \prod (x : A), P(x) \right), (f \rightsquigarrow g) \rightarrow \left( \prod (x : A) f(x) \rightsquigarrow g(x) \right).$$

**Definition 2.0.10.** Let  $f, g : \prod_{x:A} P(x)$  where  $P$  is a dependent type over  $A$ . A *homotopy from  $f$  to  $g$*  is a term  $H$  of type

$$\prod_{x:A} f(x) \rightsquigarrow_{P(x)} g(x).$$

We say that  $f$  and  $g$  are *homotopic* and denote the space of homotopies from  $f$  to  $g$  by  $f \simeq g$ .

*Remark 2.0.11.* The naive form of *functional extensionality* states that if  $f$  and  $g$  are homotopic, then they are propositionally equal.

The fact that every function preserves paths shows us, intuitively, that our type-theoretic definition of homotopy approximates our topological definition. In fact, it is just as strong in the presence of the univalence axiom. We shall see that any model of HoTT must satisfy a form of functional extensionality, which ensures that every type-theoretic homotopy induces a continuous choice of paths  $f(x) \rightsquigarrow g(x)$ , as desired.

**Definition 2.0.12.** Let  $u, v, x, y : A$ . We say that the diagram

$$\begin{array}{ccc} v & \rightsquigarrow^q & y \\ \uparrow p & & \uparrow s \\ x & \rightsquigarrow^r & u \end{array}$$

*commutes* if there is some term  $h : p * q \rightsquigarrow r * s$ , called a *witness of the commutativity*.

**Lemma 2.0.13.** Let  $f, g : A \rightarrow B$ . Suppose that  $H : f \simeq g$ . Then for each path  $p : x \rightsquigarrow_A y$ , the square

$$\begin{array}{ccc} f(y) & \rightsquigarrow^{H(y)} & g(y) \\ \uparrow f(p) & & \uparrow g(p) \\ f(x) & \rightsquigarrow^{H(x)} & g(x) \end{array}$$

*commutes*.

*Proof.* This is a straightforward application of path induction. □

The following notion corresponds to homotopy equivalence from topology.

**Definition 2.0.14.** Let  $f : A \rightarrow B$  be a function. We say that  $f$  is a (*homotopy*) *isomorphism* if there is some term  $g : B \rightarrow A$  such that  $f \circ g \simeq \text{idmap}_B$  and  $g \circ f \simeq \text{idmap}_A$ .

The space  $\text{iso}(A, B)$  of homotopy isomorphisms from  $A$  to  $B$  is precisely

$$\sum (f : A \rightarrow B) (g : B \rightarrow A), (\text{idmap}_B \simeq f \circ g) \times (\text{idmap}_A \simeq g \circ f).$$

**Proposition 2.0.15.**

1. Any inverse of  $f : A \rightarrow B$  is unique up to homotopy.
2. Any function homotopic to an isomorphism is an isomorphism (just as a continuous map in topology).

**Definition 2.0.16.** We say that a type  $A$  is *contractible* if

$$\text{isContr}(A) := \sum_{a:A} \prod_{x:A} x \rightsquigarrow a$$

is inhabited. In this case, we say that the first component of any inhabitant is the *center of contraction*.

To preserve our topological intuition, we interpret this as saying that  $A$  is contractible when there is a homotopy from  $\lambda x.x$  to  $\lambda x.a$  for some  $a : A$ .

**Example 2.0.17.** The singleton type  $\text{unit}$  is contractible. This is easy to verify by induction on  $\text{unit}$ .

### 3 Type-theoretic equivalence

We begin this section with a version of the set-theoretic inverse image.

**Definition 3.0.1.** Let  $f : A \rightarrow B$  be a function and  $b : B$ . The *homotopy fiber of  $b$*  is the space

$$\text{hFiber}(f, b) := \sum_{a:A} f(a) \rightsquigarrow b.$$

**Definition 3.0.2.** A function  $f : A \rightarrow B$  is an *equivalence from  $A$  to  $B$*  if  $\text{hFiber}(f, b)$  is contractible for each  $b : B$ , i.e., if some term has type

$$\text{isEquiv}(f) := \prod_{b:B} \text{isContr}(\text{hFiber}(f, b)).$$

In this case, we say that  $A$  and  $B$  are *equivalent* and write the space of such equivalences as  $A \simeq B$ .

Now let's define a type-theoretic analogue of the overcategory. Let  $A$  be a type with  $a : A$ . Define the dependent type  $Y(a)$  over  $A$  by

$$Y(a)(x) \equiv x \rightsquigarrow a$$

for each  $x : A$ .

**Lemma 3.0.3.** If  $p : x \rightsquigarrow_A y$ , then  $\underbrace{p \cdot p}_{w.r.t. \ Y(y)} \rightsquigarrow_{y \rightsquigarrow y} \text{refl}_y$  is inhabited.

*Proof.* This follows from another straightforward application of path induction. □

**Example 3.0.4.** For any type  $A$ , the identity map  $A \rightarrow A$  is an equivalence.



*Proof.* We must show that  $\mathbf{hFiber}(\text{idmap}_A, a)$  is contractible for each  $a : A$ . Let  $p : x \rightsquigarrow_A a$ , so that  $(x, p)$  is a canonical inhabitant of  $\mathbf{hFiber}(\text{idmap}_A, a)$ . By the path lifting lemma, we know that  $p_\Sigma(p) : (x, p) \rightsquigarrow (a, p \cdot p)$ . Lemma 3.0.3 implies that there exists a term of type  $p \cdot p \rightsquigarrow_{a \rightsquigarrow a} \text{refl}_a$ . Hence  $(x, p) \rightsquigarrow (a, \text{refl}_a)$  is inhabited, making  $(a, \text{refl}_a)$  the center of contraction.  $\square$

**Corollary 3.0.5.** *Let  $P$  be a dependent type over  $A$  and  $p : x \rightsquigarrow_A y$ . Then  $\text{transport}(p)$  is an equivalence from  $P(x)$  to  $P(y)$ .*

Our next concept is a constructive version of a unit-counit adjunction in category theory.

**Definition 3.0.6.** An *adjoint equivalence* from  $A$  to  $B$  is a 5-tuple  $(f, g, \eta, \epsilon, \alpha)$  consisting of  $f : A \rightarrow B$ ,  $g : B \rightarrow A$ ,  $\eta : \text{idmap}_A \simeq g \circ f$ ,  $\epsilon : f \circ g \simeq \text{idmap}_B$ , and witnesses  $\alpha(x)$  of the commutativity of the triangle

$$\begin{array}{ccc} f(x) & \xrightarrow{f(\eta(x))} & f(g(f(x))) \\ & \searrow \text{refl}_{f(x)} & \downarrow \epsilon(f(x)) \\ & & f(x) \end{array}$$

for each  $x : A$ .

**Theorem 3.0.7.** *A function  $f : A \rightarrow B$  is an equivalence if and only if it is an isomorphism.*

*Proof sketch.* We just consider the  $(\Leftarrow)$  direction, which is harder. We derive it from the following fact.

**Proposition 3.0.8.** *If  $(f, g, \eta, \epsilon, \alpha)$  is an adjoint equivalence from  $A$  to  $B$ , then  $f$  is an equivalence.*

Now, suppose that  $(f, g, H, K) : \text{iso}(A, B)$ . We want to construct an adjoint equivalence  $(f, g, \eta, \epsilon, \alpha)$ . Define  $\eta$  by

$$\eta(x) \equiv H(x) * g(K(f(x)))^{-1} * g(f(H(x)))^{-1}.$$

Let  $\epsilon \equiv K$ . It remains to check that  $f(\eta(x)) * \epsilon(f(x)) \rightsquigarrow \text{refl}_{f(x)}$ . To do this, we apply Lemma 2.0.13 after a bit of diagram chasing.  $\square$

**Note 3.0.9.** It is not the case that  $\text{iso}(A, B) \simeq (A \simeq B)$ .

**Corollary 3.0.10.** *Any two contractible types are equivalent.*

**Definition 3.0.11 (Extensionality).**

1. The *weak functional extensionality principle* (WFE) is that for any dependent type  $P$  over  $A$ , the space  $\prod_{x:A} P(x)$  is contractible whenever each  $P(x)$  is contractible.
2. The *strong functional extensional principle* (SFE) is that there is some term of type

$$\prod \{A : \mathbf{Type}\} \{P : A \rightarrow \mathbf{Type}\} \left( f, g : \prod_{x:A} P(x) \right), \text{isEquiv}(\mathbf{hApply}(f, g)).$$

Note that any product of contractible spaces is contractible in traditional topology.

**Proposition 3.0.12 (Axiom of choice).** *Let  $P$  be a dependent type over  $A$  and  $R : \prod_{x:A} (P(x) \rightarrow \mathbf{Type})$ . Then there is some term of type*

$$\prod_{x:A} \sum_{u:P(x)} R(x, u) \rightarrow \sum \left( s : \prod_{x:A} P(x) \right) \prod_{x:A} R(x, s(x)).$$

Let  $P$  and  $Q$  be dependent spaces over  $A$ . Suppose that  $\tau : \prod_{(x:A)} P(x) \rightarrow Q(x)$ . Define the term  $\Sigma_A \tau : \sum_{x:A} P(x) \rightarrow \sum_{x:A} Q(x)$  by  $\lambda w. (\text{pr}_1 w, \tau(\text{pr}_1 w)(\text{pr}_2 w))$ .

**Theorem 3.0.13 (Voevodsky).** *Let  $P$  and  $Q$  be dependent spaces over  $A$ . If a term  $\tau : \prod_{x:A} (P(x) \rightarrow Q(x))$  is such that  $\Sigma_A \tau$  is an equivalence, then  $\tau$  is a fiberwise equivalence, i.e.,  $\tau(x)$  is an equivalence for each  $x : A$ .*

Along with the axiom of choice and the fact that any two contractible spaces are equivalent, this result turns out to be enough to establish that  $\mathbf{WFE} \implies \mathbf{SFE}$ .

## 4 Univalence

**Lemma 4.0.1.** *For any types  $A$  and  $B$  and any path  $p : A \rightsquigarrow_{\mathbf{Type}} B$ , there is some term  $v(A, B, p) : A \simeq B$ .*

*Proof.* By path induction, it suffices to notice that the identity map on  $A$  is an equivalence.  $\square$

As a result, we obtain a function

$$v(A, B) : (A \rightsquigarrow B) \rightarrow (A \simeq B).$$

The *univalence axiom* ( $\mathbf{Univ}$ ) states that  $v(A, B)$  is an equivalence. In particular, there is some function

$$v(A, B)^{-1} : (A \simeq B) \rightarrow (A \rightsquigarrow B).$$

**Lemma 4.0.2 (Induction on equivalences).** *Suppose that*

$$D : \prod_{A, B : \mathbf{Type}} (A \simeq B) \rightarrow \mathbf{Type}$$

*and that there is some term*

$$d : \prod_{A : \mathbf{Type}} D(A, A, \text{idmap}_A).$$

*Then there is some section*

$$J(D, d) : \prod_{\substack{\{A, B : \mathbf{Type}\} \\ e : A \simeq B}} D(A, B, e)$$

*of  $D$ .*

*Proof.* Let  $A, B : \mathbf{Type}$  and  $e : A \simeq B$ . There is some path  $v(A, B)(v(A, B)^{-1}(e)) \rightsquigarrow_{A \simeq B} e$ . Thus, it suffices to show that

$$D(A, B, v(A, B)(v(A, B)^{-1}(e)))$$

is inhabited. We must find a section of the dependent type  $D'$  over  $A \rightsquigarrow_{\mathbf{Type}} B$  where  $D'(A, B, p) \equiv D(A, B, v(A, B, p))$ . But note that  $D'(A, A, \text{refl}_A) \equiv D(A, A, \text{idmap}_A)$ , which is inhabited by assumption. By path induction, we are done.  $\square$

We now have a powerful tool for proving things about function spaces.

**Corollary 4.0.3.** *If  $A \simeq B$ , then  $(X \rightarrow A) \simeq (X \rightarrow B)$  for each space  $X$ .*

**Proposition 4.0.4.** *The univalence axiom entails naive non-dependent functional extensionality.*

**Corollary 4.0.5 (Voevodsky).**  *$\mathbf{Univ}$  entails WFE (hence SFE).*

*Proof.* Suppose that  $P : A \rightarrow \mathbf{Type}$  and that there is some term  $K : \prod_{x:A} \text{isContr}(P(x))$ . We must show that  $\prod_{x:A} P(x)$  is contractible. Define  $U : A \rightarrow \mathbf{Type}$  the constant function  $\lambda x. \text{unit}$ . Since both  $P(x)$  and  $\text{unit}$  are contractible for each  $x : A$ , we have that  $P(x) \simeq \text{unit}$ . Thanks to  $\mathbf{Univ}$ , it follows that  $P(x) \rightsquigarrow \text{unit}$ . Proposition 4.0.4 now implies that  $P \rightsquigarrow_{A \rightarrow \mathbf{Type}} U$ .

This means that there is some term of type

$$\left( \prod_{x:A} P(x) \right) \rightsquigarrow \left( \prod_{x:A} U(x) \right).$$

Therefore, it suffices to show that  $\prod_{x:A} U(x)$  is contractible. But  $\prod_{x:A} U(x) \equiv A \rightarrow \text{unit}$ , and any term of this is homotopic to a constant function. Another use of Proposition 4.0.4 proves that  $A \rightarrow \text{unit}$  is contractible.  $\square$

Finally, by assuming  $\mathbf{Univ}$ , we can prove a fundamental result of category theory *in* our type theory. To this end, let  $P$  and  $Q$  be dependent types over  $A$  (viewed as type-valued presheaves on  $A$ ). Let  $\text{Hom}(P, A) := \prod_{x:A} (P(x) \rightarrow Q(x))$ .

**Lemma 4.0.6 (Yoneda).** *Assume that WFE holds. Then for any dependent type  $P$  over  $A$  and any  $a : A$ , there exists a natural equivalence*

$$\alpha_{P,a} : \text{Hom}(\mathbf{Y}(a), P) \simeq P(a).$$

*The term  $\alpha_{P,a}$  is natural in the sense that for any  $p : a \rightsquigarrow a'$  and  $\sigma : \text{Hom}(P, P')$ , the square*

$$\begin{array}{ccc} \text{Hom}(\mathbf{Y}(a), P) & \xrightarrow{\alpha_{P,a}} & P(a) \\ \text{Hom}(\mathbf{Y}(p), \sigma) \downarrow & & \downarrow \sigma(p) \\ \text{Hom}(\mathbf{Y}(a'), P') & \xrightarrow{\alpha_{P',a'}} & P'(a') \end{array}$$

*commutes up to homotopy.*<sup>1</sup>

**Corollary 4.0.7.** *By taking  $\mathbf{Y}(b)$  to be  $P$ , we have that  $\text{Hom}(\mathbf{Y}(a), \mathbf{Y}(b)) \simeq a \rightsquigarrow b$ .*

<sup>1</sup>See Egbert Rijke's proof of this at the [HoTT website](#), which resembles many standard proofs from category theory.

## 5 Models of type theory

Thus far, we have looked at the syntax of HoTT. This section is a very brief and informal summary of certain research in the semantics of HoTT.

A category modeling intensional type theory has the following structure:

$$\text{initial object} \longleftrightarrow \perp$$

$$\text{terminal object} \longleftrightarrow \text{unit}$$

$$\text{product} \longleftrightarrow P \times Q$$

$$\text{coproduct} \longleftrightarrow P + Q \quad .$$

$$\text{cartesian closed} \longleftrightarrow P \rightarrow Q$$

$$\text{locally cartesian closed} \longleftrightarrow \prod_{x:A} P(x)$$

$$\text{weak factorization system} \longleftrightarrow a \rightsquigarrow b$$

We define an interpretation function from the set of all terms of intensional type theory to such a category so that any proof in our type theory produces a theorem in the categorical structure. In other words, our categorical semantics is sound.

It is known that HoTT without **Univ** may be interpreted in the category **Set** under the theory

$$\text{ZFC} + \text{“there exist countably many inaccessible cardinals”}$$

with every type interpreted as a set. In 2006, Voevodsky showed that the category of Kan complexes is a model of HoTT. In it, a type family corresponds to a Kan fibration. His construction, however, is non-constructive to some degree as it relies heavily on the axiom of choice. Thus, it fails as a *computational* interpretation of HoTT.

Coquand, however, has designed a so-called cubical model of HoTT, in which **Univ** is a theorem. This models the  $\infty$ -groupoid structure of identity types with the category of constructive cubical sets, which are set-valued presheaves on the category of power sets of finite sets (viewed as  $n$ -cubes). Thus, Coquand’s model serves as a constructive interpretation of **Univ**. Anders Mörtberg has designed a Haskell implementation of cubical type theory, called Cubical Agda (<https://github.com/agda/cubical>).

### Further topics

One key feature of HoTT that we have neglected is the ability to define so-called higher inductive types. For example, the circle  $S^1$  can be modeled as a higher inductive type. Dan Licata even has a formalized proof

that  $\pi_1(S^1) \cong \mathbb{Z}$  in Agda.

## Sources

- [nLab](#).
- Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013.
- Altenkirch, Thorsten. *Introduction to Homotopy Type Theory*. 2017.