

# Homotopy Type Theory

## Basics

### 2.1 Types are higher groupoids

*lem]lem* For every type  $A$  and every  $x, y : A$  there is a function

$$(x = y) \rightarrow (y = x)$$

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

*lem]lem* For every type  $A$  and every  $x, y, z : A$  there is a function

$$(x = y) \rightarrow (y = z) \rightarrow (x = z)$$

written  $p \mapsto q \mapsto p \bullet q$ , such that  $\text{refl}_x \bullet \text{refl}_x \equiv \text{refl}_x$  for any  $x : A$ . We call  $p \bullet q$  the **concatenation** or **composite** of  $p$  and  $q$ .

Equality	Homotopy	$\infty$ -Groupoid
reflexivity	constant path	identity morphism
symmetry	inversion of paths	inverse morphism
transitivity	concatenation of paths	composition of morphisms

*lem]lem* Suppose  $A : \mathcal{U}$ , that  $x, y, z, w : A$  and that  $p : x = y$  and  $q : y = z$  and  $r : z = w$ . We have the following:

**Lemma 2.1.0.** (i)  $p = p \bullet \text{refl}_x$  and  $p = \text{refl}_x \bullet p$ .

(ii)  $p^{-1} \bullet p = \text{refl}_y$  and  $p \bullet p^{-1} = \text{refl}_x$ .

(iii)  $(p^{-1})^{-1} = p$ .

(iv)  $p \bullet (q \bullet r) = (p \bullet q) \bullet r$ .

**Theorem 2.1.1** (Eckmann–Hilton). *The composition operation on the second loop space*

$$\Omega^2(A) \times \Omega^2(A) \rightarrow \Omega^2(A)$$

is commutative:  $\alpha \bullet \beta = \beta \bullet \alpha$ , for any  $\alpha, \beta : \Omega^2(A)$ .

**defn]defn** A **pointed type**  $(A, a)$  is a type  $A : \mathcal{U}$  together with a point  $a : A$ , called its **basepoint**. We write  $\mathcal{U}_\bullet := \Sigma_{(A, a) \in \mathcal{U}} A$  for the type of pointed types in the universe  $\mathcal{U}$ .

**defn]defn** Given a pointed type  $(A, a)$ , we define the **loop space** of  $(A, a)$  to be the following pointed type:

$$\Omega(A, a) := ((a =_A a), \text{refl}_a).$$

An element of it will be called a **loop** at  $a$ . For  $n : \mathbb{N}$ , the  **$n$ -fold iterated loop space**  $\Omega^n(A, a)$  of a pointed type  $(A, a)$  is defined recursively by:

$$\Omega^0(A, a) := (A, a)$$

$$\Omega^{n+1}(A, a) := \Omega^n(\Omega(A, a)).$$

An element of it will be called an  **$n$ -loop** or an  **$n$ -dimensional loop** at  $a$ .

### 2.2 Functions are functors

*lem]lem* Suppose that  $f : A \rightarrow B$  is a function. Then for any  $x, y : A$  there is an operation

$$\text{ap}_f : (x =_A y) \rightarrow (f(x) =_B f(y)).$$

Moreover, for each  $x : A$  we have  $\text{ap}_f(\text{refl}_x) \equiv \text{refl}_{f(x)}$ .

The notation  $\text{ap}_f$  can be read either as the application of  $f$  to a path, or as the action on paths of  $f$ .

We note that  $\text{ap}$  behaves functorially, in all the ways that one might expect.

*lem]lem* For functions  $f : A \rightarrow B$  and  $g : B \rightarrow C$  and paths  $p : x =_A y$  and  $q : y =_B z$ , we have:

**Lemma 2.2.0.** (i)  $\text{ap}_f(p \bullet q) = \text{ap}_f(p) \bullet \text{ap}_f(q)$ .

(ii)  $\text{ap}_f(p^{-1}) = \text{ap}_f(p)^{-1}$ .

(iii)  $\text{ap}_g(\text{ap}_f(p)) = \text{ap}_{g \circ f}(p)$ .

(iv)  $\text{ap}_{\text{id}_A}(p) = p$ .

### 2.3 Type families are fibrations

*lem]lem* Suppose that  $P$  is a type family over  $A$  and that  $p : x =_A y$ . Then there is a function  $p_* : P(x) \rightarrow P(y)$ .

Sometimes, it is necessary to notate the type family  $P$  in which the transport operation happens.

$$\text{transport}^P(p, -) : P(x) \rightarrow P(y).$$

*lem]lem* Let  $P : A \rightarrow \mathcal{U}$  be a type family over  $A$  and assume we have  $u : P(x)$  for some  $x : A$ . Then for any  $p : x = y$ , we have

$$\text{lift}(u, p) : (x, u) = (y, p_*(u))$$

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

**rmk]rmk** Although we may think of a type family  $P : A \rightarrow \mathcal{U}$  as like a fibration, it is generally not a good idea to say things like “the fibration  $P : A \rightarrow \mathcal{U}$ ”, since this sounds like we are talking about a fibration with base  $\mathcal{U}$  and total space  $A$ . To repeat, when a type family  $P : A \rightarrow \mathcal{U}$  is regarded as a fibration, the base is  $A$  and the total space is  $\Sigma_{(x:A)} P(x)$ . We may also occasionally use other topological terminology when speaking about type families. For instance, we may refer to a dependent function  $f : \prod_{(x:A)} P(x)$  as a **section** of the fibration  $P$ , and we may say that something happens **fiberwise** if it happens for each  $P(x)$ . For instance, a section  $f : \prod_{(x:A)} P(x)$  shows that  $P$  is “fiberwise inhabited”.

*lem]lem* Suppose  $f : \prod_{(x:A)} P(x)$ ; then we have a map

$$\text{apd}_f : \prod_{p:x=y} (p_*(f(x)) =_{P(y)} f(y)).$$

*lem]lem* If  $P : A \rightarrow \mathcal{U}$  is defined by  $P(x) := B$  for a fixed  $B : \mathcal{U}$ , then for any  $x, y : A$  and  $p : x = y$  and  $b : B$  we have a path

$$\text{transportconst}_p^B(b) : \text{transport}^P(p, b) = b.$$

*lem]lem* For  $f : A \rightarrow B$  and  $p : x =_A y$ , we have

$$\text{apd}_f(p) = \text{transportconst}_p^B(f(x)) \bullet \text{ap}_f(p).$$

*lem]lem* Given  $P : A \rightarrow \mathcal{U}$  with  $p : x =_A y$  and  $q : y =_A z$  while  $u : P(x)$ , we have

$$q_*(p_*(u)) = (p \bullet q)_*(u).$$

*lem]lem* For a function  $f : A \rightarrow B$  and a type family  $P : B \rightarrow \mathcal{U}$ , and any  $p : x =_A y$  and  $u : P(f(x))$ , we have

$$\text{transport}^{P \circ f}(p, u) = \text{transport}^P(\text{ap}_f(p), u).$$

*lem]lem* For  $P, Q : A \rightarrow \mathcal{U}$  and a family of functions  $f : \prod_{(x:A)} P(x) \rightarrow Q(x)$ , and any  $p : x =_A y$  and  $u : P(x)$ , we have

$$\text{transport}^Q(p, f_x(u)) = f_y(\text{transport}^P(p, u)).$$

### 2.4 Homotopies and equivalences

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

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

Note that a homotopy is not the same as an identification ( $f = g$ ). However, in ?? we will introduce an axiom making homotopies and identifications “equivalent”.

The following proofs are left to the reader.

*lem]lem* Homotopy is an equivalence relation on each dependent function type  $\prod_{(x:A)} P(x)$ . That is, we have elements of the types

$$\begin{aligned} & \prod_{f: \prod_{(x:A)} P(x)} (f \sim f) \\ & \prod_{f, g: \prod_{(x:A)} P(x)} (f \sim g) \rightarrow (g \sim f) \\ & \prod_{f, g, h: \prod_{(x:A)} P(x)} (f \sim g) \rightarrow (g \sim h) \rightarrow (f \sim h). \end{aligned}$$

*lem]lem* Suppose  $H : f \sim g$  is a homotopy between functions  $f, g : A \rightarrow B$  and let  $p : x =_A y$ . Then we have

$$H(x) \bullet g(p) = f(p) \bullet H(y).$$

We may also draw this as a commutative diagram:

$$\begin{array}{ccc} f(x) & \xrightarrow{f(p)} & f(y) \\ H(x) \parallel & & \parallel H(y) \\ g(x) & \xrightarrow[g(p)]{} & g(y) \end{array}$$

**cor]cor** Let  $H : f \sim \text{id}_A$  be a homotopy, with  $f : A \rightarrow A$ . Then for any  $x : A$  we have

$$H(f(x)) = f(H(x)).$$

$$\sum_{g:B \rightarrow A} ((f \circ g \sim \text{id}_B) \times (g \circ f \sim \text{id}_A)) \quad (2.4.1)$$

defn]defn For a function  $f : A \rightarrow B$ , a **quasi-inverse** of  $f$  is a triple  $(g, \alpha, \beta)$  consisting of a function  $g : B \rightarrow A$  and homotopies  $\alpha : f \circ g \sim \text{id}_B$  and  $\beta : g \circ f \sim \text{id}_A$ .

Thus, (??) is the type of quasi-inverses of  $f$ ; we may denote it by  $\text{qinv}(f)$ .  
eg]leg For any  $p : x =_A y$  and  $z : A$ , the functions

$$(p \bullet -) : (y =_A z) \rightarrow (x =_A z) \quad \text{and} \\ (- \bullet p) : (z =_A x) \rightarrow (z =_A y)$$

have quasi-inverses given by  $(p^{-1} \bullet -)$  and  $(- \bullet p^{-1})$ , respectively;

eg]leg For any  $p : x =_A y$  and  $P : A \rightarrow \mathcal{U}$ , the function

$$\text{transport}^P(p, -) : P(x) \rightarrow P(y)$$

has a quasi-inverse given by  $\text{transport}^P(p^{-1}, -)$ ; this follows from ??.

$$(A \simeq B) := \sum_{f:A \rightarrow B} \text{isequiv}(f). \quad (2.4.2)$$

lem]lem Type equivalence is an equivalence relation on  $\mathcal{U}$ . More specifically:  
**Lemma 2.4.2.** (i) For any  $A$ , the identity function  $\text{id}_A$  is an equivalence; hence  $A \simeq A$ .

(ii) For any  $f : A \simeq B$ , we have an equivalence  $f^{-1} : B \simeq A$ .

(iii) For any  $f : A \simeq B$  and  $g : B \simeq C$ , we have  $g \circ f : A \simeq C$ .

## 2.5 The higher groupoid structure of type formers

## 2.6 Cartesian product types

$$(x =_{A \times B} y) \rightarrow (\text{pr}_1(x) =_A \text{pr}_1(y)) \times (\text{pr}_2(x) =_B \text{pr}_2(y)). \quad (2.6.1)$$

**Theorem 2.6.2.** For any  $x$  and  $y$ , the function (??) is an equivalence.

**Theorem 2.6.3.** In the above situation, we have

$$\text{transport}^{A \times B}(p, x) =_{A(w) \times B(w)} (\text{transport}^A(p, \text{pr}_1 x), \text{transport}^B(p, \text{pr}_2 x)).$$

**Theorem 2.6.4.** In the above situation, given  $x, y : A \times B$  and  $p : \text{pr}_1 x = \text{pr}_1 y$  and  $q : \text{pr}_2 x = \text{pr}_2 y$ , we have

$$f(\text{pair}^-(p, q)) =_{(f(x)=f(y))} \text{pair}^-(g(p), h(q)).$$

## 2.7 $\Sigma$ -types

**Theorem 2.7.1.** Suppose that  $P : A \rightarrow \mathcal{U}$  is a type family over a type  $A$  and let  $w, w' : \sum_{(x:A)} P(x)$ . Then there is an equivalence

$$(w = w') \simeq \sum_{(p:\text{pr}_1(w)=\text{pr}_1(w'))} p_*(\text{pr}_2(w)) = \text{pr}_2(w').$$

cor]cor For  $z : \sum_{(x:A)} P(x)$ , we have  $z = (\text{pr}_1(z), \text{pr}_2(z))$ .

**Theorem 2.7.2.** Suppose we have type families

$$P : A \rightarrow \mathcal{U} \quad \text{and} \quad Q : \left( \sum_{x:A} P(x) \right) \rightarrow \mathcal{U}.$$

Then we can construct the type family over  $A$  defined by

$$x \mapsto \sum_{u:P(x)} Q(x, u).$$

For any path  $p : x = y$  and any  $(u, z) : \sum_{(u:P(x))} Q(x, u)$  we have

$$p_*(u, z) = (p_*(u), \text{pair}^-(p, \text{refl}_{p_*(u)})_*(z)).$$

## 2.8 The unit type

**Theorem 2.8.1.** For any  $x, y : \mathbf{1}$ , we have  $(x = y) \simeq \mathbf{1}$ .

## 2.9 $\Pi$ -types and the function extensionality axiom

$$\text{happly} : (f = g) \rightarrow \prod_{x:A} (f(x) =_{B(x)} g(x)) \quad (2.9.1)$$

axiom]axiom For any  $A, B, f$ , and  $g$ , the function (??) is an equivalence.

In particular, ?? implies that (??) has a quasi-inverse

$$\text{funext} : \left( \prod_{x:A} (f(x) = g(x)) \right) \rightarrow (f = g).$$

This function is also referred to as “function extensionality”.

$$\text{refl}_f = \text{funext}(x \mapsto \text{refl}_{f(x)})$$

$$\alpha^{-1} = \text{funext}(x \mapsto \text{happly}(\alpha, x)^{-1})$$

$$\alpha \bullet \beta = \text{funext}(x \mapsto \text{happly}(\alpha, x) \bullet \text{happly}(\beta, x)).$$

Given a type  $X$ , a path  $p : x_1 =_X x_2$ , type families  $A, B : X \rightarrow \mathcal{U}$ , and a function  $f : A(x_1) \rightarrow B(x_1)$ , we have

$$\text{transport}^{A \rightarrow B}(p, f) = \left( x \mapsto \text{transport}^B(p, f(\text{transport}^A(p^{-1}, x))) \right) \quad (2.9.2)$$

where  $A \rightarrow B$  denotes abusively the type family  $X \rightarrow \mathcal{U}$  defined by

$$(A \rightarrow B)(x) := (A(x) \rightarrow B(x)).$$

Transporting dependent functions is similar, but more complicated. Suppose given  $X$  and  $p$  as before, type families  $A : X \rightarrow \mathcal{U}$  and  $B : \prod_{(x:X)} (A(x) \rightarrow \mathcal{U})$ , and also a dependent function  $f : \prod_{(a:A(x_1))} B(x_1, a)$ . Then for  $a : A(x_2)$ , we have

$$\text{transport}^{\Pi_A(B)}(p, f)(a) = \text{transport}^{\hat{B}} \left( (\text{pair}^-(p^{-1}, \text{refl}_{p^{-1}_*(a)}) \right)^{-1}, f(\text{transport}^A(p^{-1}, a)) \right)$$

where  $\Pi_A(B)$  and  $\hat{B}$  denote respectively the type families

$$\begin{aligned} \Pi_A(B) &:= (x \mapsto \prod_{(a:A(x))} B(x, a)) & : & X \rightarrow \mathcal{U} \\ \hat{B} &:= (w \mapsto B(\text{pr}_1 w, \text{pr}_2 w)) & : & (\sum_{(x:X)} A(x)) \rightarrow \mathcal{U}. \end{aligned} \quad (2.9.3)$$

lem]lem Given type families  $A, B : X \rightarrow \mathcal{U}$  and  $p : x =_X y$ , and also  $f : A(x) \rightarrow B(x)$  and  $g : A(y) \rightarrow B(y)$ , we have an equivalence

$$(p_*(f) = g) \simeq \prod_{a:A(x)} (p_*(f(a)) = g(p_*(a))).$$

Moreover, if  $q : p_*(f) = g$  corresponds under this equivalence to  $\hat{q}$ , then for  $a : A(x)$ , the path

$$\text{happly}(q, p_*(a)) : (p_*(f))(p_*(a)) = g(p_*(a))$$

is equal to the concatenated path  $i \bullet j \bullet k$ , where

**Lemma 2.9.3.** •  $i : (p_*(f))(p_*(a)) = p_*(f(p^{-1}_*(p_*(a))))$  comes from (??),  
•  $j : p_*(f(p^{-1}_*(p_*(a)))) = p_*(f(a))$  comes from ????, and  
•  $k : p_*(f(a)) = g(p_*(a))$  is  $\hat{q}(a)$ .

lem]lem Given type families  $A : X \rightarrow \mathcal{U}$  and  $B : \prod_{(x:X)} A(x) \rightarrow \mathcal{U}$  and  $p : x =_X y$ , and also  $f : \prod_{(a:A(x))} B(x, a)$  and  $g : \prod_{(a:A(y))} B(y, a)$ , we have an equivalence

$$(p_*(f) = g) \simeq \left( \prod_{a:A(x)} \text{transport}^{\hat{B}}(\text{pair}^-(p, \text{refl}_{p_*(a)}), f(a)) = g(p_*(a)) \right)$$

with  $\hat{B}$  as in (??).

## 2.10 Universes and the univalence axiom

lem]lem For types  $A, B : \mathcal{U}$ , there is a certain function,

$$\text{idtoeqv} : (A =_{\mathcal{U}} B) \rightarrow (A \simeq B), \quad (2.10.1)$$

defined in the proof.

axiom]axiom For any  $A, B : \mathcal{U}$ , the function (??) is an equivalence.

- An introduction rule for  $(A =_{\mathcal{U}} B)$ , denoted  $\text{ua}$  for “univalence axiom”:  
$$\text{ua} : (A \simeq B) \rightarrow (A =_{\mathcal{U}} B).$$
- The elimination rule, which is  $\text{idtoeqv}$ ,  
$$\text{idtoeqv} \equiv \text{transport}^{X \rightarrow X} : (A =_{\mathcal{U}} B) \rightarrow (A \simeq B).$$
- The propositional computation rule,  
$$\text{transport}^{X \rightarrow X}(\text{ua}(f), x) = f(x).$$
- The propositional uniqueness principle: for any  $p : A = B$ ,  
$$p = \text{ua}(\text{transport}^{X \rightarrow X}(p)).$$

We can also identify the reflexivity, concatenation, and inverses of equalities in the universe with the corresponding operations on equivalences:

$$\begin{aligned} \text{refl}_A &= \text{ua}(\text{id}_A) \\ \text{ua}(f) \bullet \text{ua}(g) &= \text{ua}(g \circ f) \\ \text{ua}(f)^{-1} &= \text{ua}(f^{-1}). \end{aligned}$$

lem]lem For any type family  $B : A \rightarrow \mathcal{U}$  and  $x, y : A$  with a path  $p : x = y$  and  $u : B(x)$ , we have

$$\begin{aligned} \text{transport}^B(p, u) &= \text{transport}^{X \rightarrow X}(\text{ap}_B(p), u) \\ &= \text{idtoeqv}(\text{ap}_B(p))(u). \end{aligned}$$

## 2.11 Identity type

**Theorem 2.11.1.** *If  $f : A \rightarrow B$  is an equivalence, then for all  $a, a' : A$ , so is*

$$\mathsf{ap}_f : (a =_A a') \rightarrow (f(a) =_B f(a')).$$

*lem* For any  $A$  and  $a : A$ , with  $p : x_1 = x_2$ , we have

$$\begin{aligned} \mathsf{transport}^{x \mapsto (a=x)}(p, q) &= q \bullet p && \text{for } q : a = x_1, \\ \mathsf{transport}^{x \mapsto (x=a)}(p, q) &= p^{-1} \bullet q && \text{for } q : x_1 = a, \\ \mathsf{transport}^{x \mapsto (x=x)}(p, q) &= p^{-1} \bullet q \bullet p && \text{for } q : x_1 = x_1. \end{aligned}$$

**Theorem 2.11.2.** *For  $f, g : A \rightarrow B$ , with  $p : a =_A a'$  and  $q : f(a) =_B g(a)$ , we have*

$$\mathsf{transport}^{x \mapsto f(x)=Bg(x)}(p, q) =_{f(a')=g(a')} (\mathsf{ap}_f p)^{-1} \bullet q \bullet \mathsf{ap}_g p.$$

**Theorem 2.11.3.** *Let  $B : A \rightarrow \mathcal{U}$  and  $f, g : \prod_{(x:A)} B(x)$ , with  $p : a =_A a'$  and  $q : f(a) =_{B(a)} g(a)$ . Then we have*

$$\mathsf{transport}^{x \mapsto f(x)=Bg(x)}(p, q) = (\mathsf{apd}_f(p))^{-1} \bullet \mathsf{ap}_{(\mathsf{transport}^B p)}(q) \bullet \mathsf{apd}_g(p).$$

**Theorem 2.11.4.** *For  $p : a =_A a'$  with  $q : a = a$  and  $r : a' = a'$ , we have*

$$(\mathsf{transport}^{x \mapsto (x=x)}(p, q) = r) \simeq (q \bullet p = p \bullet r).$$

## 2.12 Coproducts

**Theorem 2.12.1.** *For all  $x : A + B$  we have  $(\mathsf{inl}(a_0) = x) \simeq \mathsf{code}(x)$ .*

## 2.13 Natural numbers

We use the encode-decode method to characterize the path space of the natural numbers, which are also a positive type.

$$\mathsf{code} : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathcal{U},$$

defined by double recursion over  $\mathbb{N}$  as follows:

$$\begin{aligned} \mathsf{code}(0, 0) &\equiv \mathbf{1} \\ \mathsf{code}(\mathsf{succ}(m), 0) &\equiv \mathbf{0} \\ \mathsf{code}(0, \mathsf{succ}(n)) &\equiv \mathbf{0} \\ \mathsf{code}(\mathsf{succ}(m), \mathsf{succ}(n)) &\equiv \mathsf{code}(m, n). \end{aligned}$$

We also define by recursion a dependent function  $r : \prod_{(n:\mathbb{N})} \mathsf{code}(n, n)$ , with

$$\begin{aligned} r(0) &\equiv \star \\ r(\mathsf{succ}(n)) &\equiv r(n). \end{aligned}$$

**Theorem 2.13.1.** *For all  $m, n : \mathbb{N}$  we have  $(m = n) \simeq \mathsf{code}(m, n)$ .*

## 2.14 Example: equality of structures

defn]defnGiven a type  $A$ , the type  $\mathsf{SemigroupStr}(A)$  of **semigroup structures** with carrier  $A$  is defined by

$$\mathsf{SemigroupStr}(A) \equiv \sum_{(m:A \rightarrow A \rightarrow A)} \prod_{(x,y,z:A)} m(x, m(y, z)) = m(m(x, y), z).$$

A **semigroup** is a type together with such a structure:

$$\mathsf{Semigroup} \equiv \sum_{A:\mathcal{U}} \mathsf{SemigroupStr}(A)$$

### 2.14.1 Lifting equivalences

$$\mathsf{transport}^{\mathsf{SemigroupStr}}(\mathsf{ua}(e)) : \mathsf{SemigroupStr}(A) \rightarrow \mathsf{SemigroupStr}(B).$$

Moreover, this map is an equivalence, because  $\mathsf{transport}^C(\alpha)$  is always an equivalence with inverse  $\mathsf{transport}^C(\alpha^{-1})$ , see ????

## 2.15 Universal properties

$$(X \rightarrow A \times B) \rightarrow (X \rightarrow A) \times (X \rightarrow B) \quad (2.15.1)$$

defined by  $f \mapsto (\mathsf{pr}_1 \circ f, \mathsf{pr}_2 \circ f)$ .

**Theorem 2.15.2.** *(??) is an equivalence.*

$$\left( \prod_{x:X} (A(x) \times B(x)) \right) \rightarrow \left( \prod_{x:X} A(x) \right) \times \left( \prod_{x:X} B(x) \right) \quad (2.15.3)$$

defined as before by  $f \mapsto (\mathsf{pr}_1 \circ f, \mathsf{pr}_2 \circ f)$ .

**Theorem 2.15.4.** *(??) is an equivalence.*

$$\left( \prod_{x:X} \sum_{(a:A(x))} P(x, a) \right) \rightarrow \left( \sum_{(g:\prod_{(x:X)} A(x))} \prod_{(x:X)} P(x, g(x)) \right). \quad (2.15.5)$$

**Theorem 2.15.6.** *(??) is an equivalence.*

# Homotopy Type Theory

## Sets and logic

### 3.16 Sets and $n$ -types

defn]defn A type  $A$  is a **set** if for all  $x, y : A$  and all  $p, q : x = y$ , we have  $p = q$ .

More precisely, the proposition  $\text{isSet}(A)$  is defined to be the type

$$\text{isSet}(A) := \prod_{(x,y:A)} \prod_{(p,q:x=y)} (p = q).$$

eg]leg The type  $\mathbf{1}$  is a set. For any  $x, y : \mathbf{1}$  the type  $(x = y)$  is equivalent to  $\mathbf{1}$ . Since any two elements of  $\mathbf{1}$  are equal, this implies that any two elements of  $x = y$  are equal.

eg]leg The type  $\mathbf{0}$  is a set, for given any  $x, y : \mathbf{0}$  we may deduce anything we like, by the induction principle of  $\mathbf{0}$ .

eg]leg The type  $\mathbb{N}$  of natural numbers is also a set. Since all equality types  $x =_{\mathbb{N}} y$  are equivalent to either  $\mathbf{1}$  or  $\mathbf{0}$ , and any two inhabitants of  $\mathbf{1}$  or  $\mathbf{0}$  are equal.

Most of the type forming operations we have considered so far also preserve sets.

eg]leg If  $A$  and  $B$  are sets, then so is  $A \times B$ . For given  $x, y : A \times B$  and  $p, q : x = y$ , then we have  $p = \text{pair}^{\neg}(\text{ap}_{\text{pr}_1}(p), \text{ap}_{\text{pr}_2}(p))$  and  $q = \text{pair}^{\neg}(\text{ap}_{\text{pr}_1}(q), \text{ap}_{\text{pr}_2}(q))$ . But  $\text{ap}_{\text{pr}_1}(p) = \text{ap}_{\text{pr}_1}(q)$  since  $A$  is a set, and  $\text{ap}_{\text{pr}_2}(p) = \text{ap}_{\text{pr}_2}(q)$  since  $B$  is a set; hence  $p = q$ . Similarly, if  $A$  is a set and  $B : A \rightarrow \mathcal{U}$  is such that each  $B(x)$  is a set, then  $\sum_{(x:A)} B(x)$  is a set.

eg]leg If  $A$  is any type and  $B : A \rightarrow \mathcal{U}$  is such that each  $B(x)$  is a set, then the type  $\prod_{(x:A)} B(x)$  is a set. For suppose  $f, g : \prod_{(x:A)} B(x)$  and  $p, q : f = g$ . By function extensionality, we have

$$p = \text{funext}(x \mapsto \text{happly}(p, x)) \quad \text{and} \quad q = \text{funext}(x \mapsto \text{happly}(q, x)).$$

But for any  $x : A$ , we have

$$\text{happly}(p, x) : f(x) = g(x) \quad \text{and} \quad \text{happly}(q, x) : f(x) = g(x),$$

so since  $B(x)$  is a set we have  $\text{happly}(p, x) = \text{happly}(q, x)$ . Now using function extensionality again, the dependent functions  $(x \mapsto \text{happly}(p, x))$  and  $(x \mapsto \text{happly}(q, x))$  are equal, and hence (applying  $\text{ap}_{\text{funext}}$ ) so are  $p$  and  $q$ .

defn]defn A type  $A$  is a **1-type** if for all  $x, y : A$  and  $p, q : x = y$  and  $r, s : p = q$ , we have  $r = s$ .

lem]lem If  $A$  is a set (that is,  $\text{isSet}(A)$  is inhabited), then  $A$  is a 1-type.

### 3.17 Propositions as types

rmk]rmk (Statement) If for all  $x : X$  there exists an  $a : A(x)$  such that  $P(x, a)$ , then there exists a function  $g : \prod_{(x:A)} A(x)$  such that for all  $x : X$  we have  $P(x, g(x))$ . This looks like the classical *axiom of choice*, is always true under this reading.

rmk]rmk The classical *law of double negation* and *law of excluded middle* are incompatible with the univalence axiom.

**Theorem 3.17.1.** *It is not the case that for all  $A : \mathcal{U}$  we have  $\neg(\neg A) \rightarrow A$ .*

rmk]rmk For any  $A$ ,  $\neg\neg\neg A \rightarrow \neg A$  for any  $A$ .

cor]cor It is not the case that for all  $A : \mathcal{U}$  we have  $A + (\neg A)$ .

### 3.18 Mere propositions

defn]defn A type  $P$  is a **mere proposition** if for all  $x, y : P$  we have  $x = y$ .

Specifically, for any  $P : \mathcal{U}$ , the type  $\text{isProp}(P)$  is defined to be

$$\text{isProp}(P) := \prod_{x,y:P} (x = y).$$

lem]lem If  $P$  is a mere proposition and  $x_0 : P$ , then  $P \simeq \mathbf{1}$ .

lem]lem If  $P$  and  $Q$  are mere propositions such that  $P \rightarrow Q$  and  $Q \rightarrow P$ , then  $P \simeq Q$ .

lem]lem Every mere proposition is a set.

lem]lem For any type  $A$ , the types  $\text{isProp}(A)$  and  $\text{isSet}(A)$  are mere propositions.

### 3.19 Classical vs. intuitionistic logic

With the notion of mere proposition in hand, we can now give the proper formulation of the **law of excluded middle** in homotopy type theory:

$$\text{LEM} := \prod_{A:\mathcal{U}} (\text{isProp}(A) \rightarrow (A + \neg A)). \quad (3.19.1)$$

Similarly, the **law of double negation** is

$$\prod_{A:\mathcal{U}} (\text{isProp}(A) \rightarrow (\neg\neg A \rightarrow A)). \quad (3.19.2)$$

- defn]defn
- Definition 3.19.2.** (i) A type  $A$  is called **decidable** if  $A + \neg A$ .  
 (ii) Similarly, a type family  $B : A \rightarrow \mathcal{U}$  is **decidable** if  $\prod_{(a:A)} (B(a) + \neg B(a))$ .  
 (iii) In particular,  $A$  has **decidable equality** if  $\prod_{(a,b:A)} ((a = b) + \neg(a = b))$ .

### 3.20 Subsets and propositional resizing

lem]lem Suppose  $P : A \rightarrow \mathcal{U}$  is a type family such that  $P(x)$  is a mere proposition for all  $x : A$ . If  $u, v : \sum_{(x:A)} P(x)$  are such that  $\text{pr}_1(u) = \text{pr}_1(v)$ , then  $u = v$ .

For instance, recall that

$$(A \simeq B) := \sum_{f:A \rightarrow B} \text{isequiv}(f),$$

where each type  $\text{isequiv}(f)$  was supposed to be a mere proposition. It follows that if two equivalences have equal underlying functions, then they are equal as equivalences.

If  $P : A \rightarrow \mathcal{U}$  is a family of mere propositions (i.e. each  $P(x)$  is a mere proposition), we may write

$$\{x : A \mid P(x)\} \quad (3.20.1)$$

as an alternative notation for  $\sum_{(x:A)} P(x)$ . We may define the “subuniverses” of sets and of mere propositions in a universe  $\mathcal{U}$ :

$$\begin{aligned} \text{Set}_{\mathcal{U}} &:= \{A : \mathcal{U} \mid \text{isSet}(A)\}, \\ \text{Prop}_{\mathcal{U}} &:= \{A : \mathcal{U} \mid \text{isProp}(A)\}. \end{aligned}$$

An element of  $\text{Set}_{\mathcal{U}}$  is a type  $A : \mathcal{U}$  together with evidence  $s : \text{isSet}(A)$ , and similarly for  $\text{Prop}_{\mathcal{U}}$ .

axiom]axiom The map  $\text{Prop}_{\mathcal{U}_i} \rightarrow \text{Prop}_{\mathcal{U}_{i+1}}$  is an equivalence.

With propositional resizing, we can define the power set to be

$$\mathcal{P}(A) := (A \rightarrow \Omega),$$

which is then independent of  $\mathcal{U}$ .

### 3.21 The logic of mere propositions

eg]leg If  $A$  and  $B$  are mere propositions, so is  $A \times B$ . This is easy to show using the characterization of paths in products, just like ?? but simpler. Thus, the connective “and” preserves mere propositions.

eg]leg If  $A$  is any type and  $B : A \rightarrow \mathcal{U}$  is such that for all  $x : A$ , the type  $B(x)$  is a mere proposition, then  $\prod_{(x:A)} B(x)$  is a mere proposition. The proof is just like ?? but simpler: given  $f, g : \prod_{(x:A)} B(x)$ , for any  $x : A$  we have  $f(x) = g(x)$  since  $B(x)$  is a mere proposition. But then by function extensionality, we have  $f = g$ . In particular, if  $B$  is a mere proposition, then so is  $A \rightarrow B$  regardless of what  $A$  is. In even more particular, since  $\mathbf{0}$  is a mere proposition, so is  $\neg A \equiv (A \rightarrow \mathbf{0})$ . Thus, the connectives “implies” and “not” preserve mere propositions, as does the quantifier “for all”.

### 3.22 Propositional truncation

The *propositional truncation*, also called the  $(-1)$ -truncation, *bracket type*, or *squash type*, is an additional type former which “squashes” or “truncates” a type down to a mere proposition, forgetting all information contained in inhabitants of that type other than their existence. More precisely, for any type  $A$ , there is a type  $\|A\|$ . It has two constructors:

- For any  $a : A$  we have  $|a| : \|A\|$ .
- For any  $x, y : \|A\|$ , we have  $x = y$ .

The recursion principle of  $\|A\|$  says that:

- If  $B$  is a mere proposition and we have  $f : A \rightarrow B$ , then there is an induced  $g : \|A\| \rightarrow B$  such that  $g(|a|) \equiv f(a)$  for all  $a : A$ .

We define **traditional logical notation** using truncation as follows, where  $P$  and  $Q$  denote mere propositions (or families thereof):

$$\begin{aligned}\top &::= \mathbf{1} \\ \perp &::= \mathbf{0} \\ P \wedge Q &::= P \times Q \\ P \Rightarrow Q &::= P \rightarrow Q \\ P \Leftrightarrow Q &::= P = Q \\ \neg P &::= P \rightarrow \mathbf{0} \\ P \vee Q &::= \|P + Q\| \\ \forall(x : A). P(x) &::= \prod_{x:A} P(x) \\ \exists(x : A). P(x) &::= \left\| \sum_{x:A} P(x) \right\|\end{aligned}$$

The notations  $\wedge$  and  $\vee$  are also used in homotopy theory for the smash product and the wedge of pointed spaces.

$$\begin{aligned}\{x : A \mid P(x)\} \cap \{x : A \mid Q(x)\} &::= \{x : A \mid P(x) \wedge Q(x)\}, \\ \{x : A \mid P(x)\} \cup \{x : A \mid Q(x)\} &::= \{x : A \mid P(x) \vee Q(x)\}, \\ A \setminus \{x : A \mid P(x)\} &::= \{x : A \mid \neg P(x)\}.\end{aligned}$$

Of course, in the absence of LEM, the latter are not “complements” in the usual sense: we may not have  $B \cup (A \setminus B) = A$  for every subset  $B$  of  $A$ .

### 3.23 The axiom of choice

$$A : X \rightarrow \mathcal{U} \quad \text{and} \quad P : \prod_{x:X} A(x) \rightarrow \mathcal{U},$$

and moreover that

- $X$  is a set,
- $A(x)$  is a set for all  $x : X$ , and
- $P(x, a)$  is a mere proposition for all  $x : X$  and  $a : A(x)$ .

The **axiom of choice** AC asserts that under these assumptions,

$$\left( \prod_{x:X} \left\| \sum_{a:A(x)} P(x, a) \right\| \right) \rightarrow \left\| \sum_{(g:\prod_{(x:X)} A(x))} \prod_{(x:X)} P(x, g(x)) \right\|. \quad (3.23.1)$$

Of course, this is a direct translation of (??) where we read “there exists  $x : A$  such that  $B(x)$ ” as  $\left\| \sum_{(x:A)} B(x) \right\|$ , so we could have written the statement in the familiar logical notation as

$$\left( \forall(x : X). \exists(a : A(x)). P(x, a) \right) \Rightarrow \left( \exists(g : \prod_{(x:X)} A(x)). \forall(x : X). P(x, g(x)) \right).$$

*lem*]lem The axiom of choice (??) is equivalent to the statement that for any set  $X$  and any  $Y : X \rightarrow \mathcal{U}$  such that each  $Y(x)$  is a set, we have

$$\left( \prod_{x:X} \left\| Y(x) \right\| \right) \rightarrow \left\| \prod_{x:X} Y(x) \right\|. \quad (3.23.2)$$

*rmk*]rmk The right side of (??) always implies the left. Since both are mere propositions, by ?? the axiom of choice is also equivalent to asking for an equivalence

$$\left( \prod_{x:X} \left\| Y(x) \right\| \right) \simeq \left\| \prod_{x:X} Y(x) \right\|$$

*lem*]lem There exists a type  $X$  and a family  $Y : X \rightarrow \mathcal{U}$  such that each  $Y(x)$  is a set, but such that (??) is false.

### 3.24 The principle of unique choice

*lem*]lem If  $P$  is a mere proposition, then  $P \simeq \|P\|$ .

*cor*]cor Suppose a type family  $P : A \rightarrow \mathcal{U}$  such that

- Corollary 3.24.0** (The principle of unique choice). (i) For each  $x$ , the type  $P(x)$  is a mere proposition, and
- (ii) For each  $x$  we have  $\|P(x)\|$ .
- Then we have  $\prod_{(x:A)} P(x)$ .

### 3.26 Contractibility

A type  $A$  is **contractible**, or a **singleton**, if there is  $a : A$ , called the **center of contraction**, such that  $a = x$  for all  $x : A$ . We denote the specified path  $a = x$  by  $\text{contr}_x$ .

In other words, the type  $\text{isContr}(A)$  is defined to be

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

*lem*]lem For a type  $A$ , the following are logically equivalent.

**Lemma 3.26.0.** (i)  $A$  is contractible in the sense of ??.

(ii)  $A$  is a mere proposition, and there is a point  $a : A$ .

(iii)  $A$  is equivalent to  $\mathbf{1}$ .

*lem*]lem For any type  $A$ , the type  $\text{isContr}(A)$  is a mere proposition.

*cor*]cor If  $A$  is contractible, then so is  $\text{isContr}(A)$ .

*lem*]lem If  $P : A \rightarrow \mathcal{U}$  is a type family such that each  $P(a)$  is contractible, then  $\prod_{(x:A)} P(x)$  is contractible.

Of course, if  $A$  is equivalent to  $B$  and  $A$  is contractible, then so is  $B$ . More generally, it suffices for  $B$  to be a *retract* of  $A$ . By definition, a **retraction** is a function  $r : A \rightarrow B$  such that there exists a function  $s : B \rightarrow A$ , called its **section**, and a homotopy  $e : \prod_{(y:B)} (r(s(y)) = y)$ ; then we say that  $B$  is a **retract** of  $A$ .

*lem*]lem If  $B$  is a retract of  $A$ , and  $A$  is contractible, then so is  $B$ .

*lem*]lem For any  $A$  and any  $a : A$ , the type  $\sum_{(x:A)} (a = x)$  is contractible.

*lem*]lem Let  $P : A \rightarrow \mathcal{U}$  be a type family.

- Lemma 3.26.0.** (i) If each  $P(x)$  is contractible, then  $\sum_{(x:A)} P(x)$  is equivalent to  $A$ .
- (ii) If  $A$  is contractible with center  $a$ , then  $\sum_{(x:A)} P(x)$  is equivalent to  $P(a)$ .

*lem*]lem A type  $A$  is a mere proposition if and only if for all  $x, y : A$ , the type  $x =_A y$  is contractible.