# Homotopy Type Theory

Basics

## 2.1 Types are higher groupoids

**Lemma 2.1.1.** *For every type $A$ and every $x, y : A$ there is a function*

$$(x = y) \to (y = x)$$

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

**Lemma 2.1.2.** *For every type $A$ and every $x, y, z : A$ there is a function*

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

*written $p \mapsto q \mapsto p \cdot q$, such that $\mathsf{refl}_x \cdot \mathsf{refl}_x \equiv \mathsf{refl}_x$ for any $x : A$. We call $p \cdot q$ 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 |

**Lemma 2.1.3.** *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:*

*(i).* $p = p \cdot \mathsf{refl}_y$ *and* $p = \mathsf{refl}_x \cdot p$.
*(ii).* $p^{-1} \cdot p = \mathsf{refl}_y$ *and* $p \cdot p^{-1} = \mathsf{refl}_x$.
*(iii).* $(p^{-1})^{-1} = p$.
*(iv).* $p \cdot (q \cdot r) = (p \cdot q) \cdot r$.

**Theorem 2.1.4** (Eckmann–Hilton)**.** *The composition operation on the second loop space*

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

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

**Definition 2.1.5.** A **pointed type** $(A, a)$ is a type $A : \mathcal{U}$ together with a point $a : A$, called its **basepoint**. We write $\mathcal{U}_\bullet :\equiv \sum_{(A:\mathcal{U})} A$ for the type of pointed types in the universe $\mathcal{U}$.

**Definition 2.1.6.** Given a pointed type $(A, a)$, we define the **loop space** of $(A, a)$ to be the following pointed type:

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

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:

$$\Omega^0(A, a) :\equiv (A, a)$$
$$\Omega^{n+1}(A, a) :\equiv \Omega^n(\Omega(A, a)).$$

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

## 2.2 Functions are functors

**Lemma 2.2.1.** *Suppose that $f : A \to B$ is a function. Then for any $x, y : A$ there is an operation*

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

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

The notation $\mathsf{ap}_f$ can be read either as the $\underline{\text{ap}}$plication of $f$ to a path, or as the $\underline{\text{a}}$ction on $\underline{\text{p}}$aths of $f$.
We note that $\mathsf{ap}$ behaves functorially, in all the ways that one might expect.

**Lemma 2.2.2.** *For functions $f : A \to B$ and $g : B \to C$ and paths $p : x =_A y$ and $q : y =_A z$, we have:*

*(i).* $\mathsf{ap}_f(p \cdot q) = \mathsf{ap}_f(p) \cdot \mathsf{ap}_f(q)$.
*(ii).* $\mathsf{ap}_f(p^{-1}) = \mathsf{ap}_f(p)^{-1}$.
*(iii).* $\mathsf{ap}_g(\mathsf{ap}_f(p)) = \mathsf{ap}_{g \circ f}(p)$.
*(iv).* $\mathsf{ap}_{\mathsf{id}_A}(p) = p$.

## 2.3 Type families are fibrations

**Lemma 2.3.1** (Transport)**.** *Suppose that $P$ is a type family over $A$ and that $p : x =_A y$. Then there is a function $p_* : P(x) \to P(y)$.*

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

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

**Lemma 2.3.2** (Path lifting property)**.** *Let $P : A \to \mathcal{U}$ 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*

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

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

*Remark* 2.3.3. Although we may think of a type family $P : A \to \mathcal{U}$ as like a fibration, it is generally not a good idea to say things like "the fibration $P : A \to \mathcal{U}$", since this sounds like we are talking about a fibration with base $\mathcal{U}$ and total space $A$. To repeat, when a type family $P : A \to \mathcal{U}$ is regarded as a fibration, the base is $A$ and the total space is $\sum_{(x:A)} P(x)$.
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".

**Lemma 2.3.4** (Dependent map)**.** *Suppose $f : \prod_{(x:A)} P(x)$; then we have a map*

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

**Lemma 2.3.5.** *If $P : A \to \mathcal{U}$ is defined by $P(x) :\equiv B$ for a fixed $B : \mathcal{U}$, then for any $x, y : A$ and $p : x = y$ and $b : B$ we have a path*

$$\mathsf{transportconst}^B_p(b) : \mathsf{transport}^P(p, b) = b.$$

**Lemma 2.3.8.** *For $f : A \to B$ and $p : x =_A y$, we have*

$$\mathsf{apd}_f(p) = \mathsf{transportconst}^B_p(f(x)) \cdot \mathsf{ap}_f(p).$$

**Lemma 2.3.9.** *Given $P : A \to \mathcal{U}$ with $p : x =_A y$ and $q : y =_A z$ while $u : P(x)$, we have*

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

**Lemma 2.3.10.** *For a function $f : A \to B$ and a type family $P : B \to \mathcal{U}$, and any $p : x =_A y$ and $u : P(f(x))$, we have*

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

**Lemma 2.3.11.** *For $P, Q : A \to \mathcal{U}$ and a family of functions $f : \prod_{(x:A)} P(x) \to Q(x)$, and any $p : x =_A y$ and $u : P(x)$, we have*

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

## 2.4 Homotopies and equivalences

**Definition 2.4.1.** Let $f, g : \prod_{(x:A)} P(x)$ be two sections of a type family $P : A \to \mathcal{U}$. A **homotopy** from $f$ to $g$ is a dependent function of type

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

Note that a homotopy is not the same as an identification ($f = g$). However, in §2.9 we will introduce an axiom making homotopies and identifications "equivalent".
The following proofs are left to the reader.

**Lemma 2.4.2.** *Homotopy is an equivalence relation on each dependent function type $\prod_{(x:A)} P(x)$. That is, we have elements of the types*

$$\prod_{f:\prod_{(x:A)} P(x)} (f \sim f)$$
$$\prod_{f,g:\prod_{(x:A)} P(x)} (f \sim g) \to (g \sim f)$$
$$\prod_{f,g,h:\prod_{(x:A)} P(x)} (f \sim g) \to (g \sim h) \to (f \sim h).$$

**Lemma 2.4.4.** *Suppose $H : f \sim g$ is a homotopy between functions $f, g : A \to B$ and let $p : x =_A y$. Then we have*

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

*We may also draw this as a commutative diagram:*

$$\begin{tikzcd}
f(x) \arrow[r, equal, "f(p)"] \arrow[d, equal, "H(x)"'] & f(y) \arrow[d, equal, "H(y)"] \\
g(x) \arrow[r, equal, "g(p)"'] & g(y)
\end{tikzcd}$$

**Corollary 2.4.5.** *Let $H : f \sim \mathsf{id}_A$ be a homotopy, with $f : A \to A$. Then for any $x : A$ we have*
$$H(f(x)) = f(H(x)).$$

$$\sum_{g:B\to A} \big((f \circ g \sim \mathsf{id}_B) \times (g \circ f \sim \mathsf{id}_A)\big) \tag{2.4.6}$$

**Definition 2.4.7.** For a function $f : A \to B$, a **quasi-inverse** of $f$ is a triple $(g, \alpha, \beta)$ consisting of a function $g : B \to A$ and homotopies $\alpha : f \circ g \sim \mathsf{id}_B$ and $\beta : g \circ f \sim \mathsf{id}_A$.

Thus, (2.4.6) is *the type of quasi-inverses of $f$*; we may denote it by $\mathsf{qinv}(f)$.

*Example* 2.4.8. For any $p : x =_A y$ and $z : A$, the functions
$$\begin{aligned}(p \cdot -) &: (y =_A z) \to (x =_A z) \qquad \text{and}\\ (- \cdot p) &: (z =_A x) \to (z =_A y)\end{aligned}$$
have quasi-inverses given by $(p^{-1} \cdot -)$ and $(- \cdot p^{-1})$, respectively;

*Example* 2.4.9. For any $p : x =_A y$ and $P : A \to \mathcal{U}$, the function
$$\mathsf{transport}^P(p, -) : P(x) \to P(y)$$
has a quasi-inverse given by $\mathsf{transport}^P(p^{-1}, -)$; this follows from Lemma 2.3.9.

$$\mathsf{isequiv}(f) :\equiv \Big(\sum_{g:B\to A} (f \circ g \sim \mathsf{id}_B)\Big) \times \Big(\sum_{h:B\to A} (h \circ f \sim \mathsf{id}_A)\Big). \tag{2.4.10}$$

$$(A \simeq B) :\equiv \sum_{f:A\to B} \mathsf{isequiv}(f). \tag{2.4.11}$$

**Lemma 2.4.12.** *Type equivalence is an equivalence relation on $\mathcal{U}$. More specifically:*

*(i). For any $A$, the identity function $\mathsf{id}_A$ 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

$$(x =_{A\times B} y) \to (\mathsf{pr}_1(x) =_A \mathsf{pr}_1(y)) \times (\mathsf{pr}_2(x) =_B \mathsf{pr}_2(y)). \tag{2.6.1}$$

**Theorem 2.6.2.** *For any $x$ and $y$, the function (2.6.1) is an equivalence.*

**Theorem 2.6.3.** *In the above situation, we have*
$$\mathsf{transport}^{A\times B}(p, x) =_{A(w)\times B(w)} (\mathsf{transport}^A(p, \mathsf{pr}_1 x), \mathsf{transport}^B(p, \mathsf{pr}_2 x)).$$

**Theorem 2.6.4.** *In the above situation, given $x, y : A \times B$ and $p : \mathsf{pr}_1 x = \mathsf{pr}_1 y$ and $q : \mathsf{pr}_2 x = \mathsf{pr}_2 y$, we have*
$$f(\mathsf{pair}^=(p, q)) =_{(f(x)=f(y))} \mathsf{pair}^=(g(p), h(q)).$$

## 2.7 Σ-types

**Theorem 2.7.2.** *Suppose that $P : A \to \mathcal{U}$ is a type family over a type $A$ and let $w, w' : \sum_{(x:A)} P(x)$. Then there is an equivalence*
$$(w = w') \simeq \sum_{(p:\mathsf{pr}_1(w)=\mathsf{pr}_1(w'))} p_*(\mathsf{pr}_2(w)) = \mathsf{pr}_2(w').$$

**Corollary 2.7.3.** *For $z : \sum_{(x:A)} P(x)$, we have $z = (\mathsf{pr}_1(z), \mathsf{pr}_2(z))$.*

Note that the lifted path $\mathsf{lift}(u, p)$ of $p : x = y$ at $u : P(x)$ defined in Lemma 2.3.2 may be identified with the special case of the introduction form
$$\mathsf{pair}^=(p, \mathsf{refl}_{p_*(u)}) : (x, u) = (y, p_*(u)).$$

**Theorem 2.7.4.** *Suppose we have type families*
$$P : A \to \mathcal{U} \qquad \text{and} \qquad Q : \Big(\sum_{x:A} P(x)\Big) \to \mathcal{U}.$$
*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*
$$p_*((u, z)) = \big(p_*(u), \mathsf{pair}^=(p, \mathsf{refl}_{p_*(u)})_*(z)\big).$$

## 2.8 The unit type

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

## 2.9 Π-types and the function extensionality axiom

Given a type $A$ and a type family $B : A \to \mathcal{U}$, consider the dependent function type $\prod_{(x:A)} B(x)$. We expect the type $f = g$ of paths from $f$ to $g$ in $\prod_{(x:A)} B(x)$ to be equivalent to the type of pointwise paths:
$$(f = g) \simeq \Big(\prod_{x:A} (f(x) =_{B(x)} g(x))\Big). \tag{2.9.1}$$

$$\mathsf{happly} : (f = g) \to \prod_{x:A} (f(x) =_{B(x)} g(x)) \tag{2.9.2}$$

**Axiom 2.9.3** (Function extensionality). *For any $A$, $B$, $f$, and $g$, the function (2.9.2) is an equivalence.*

In particular, Axiom 2.9.3 implies that (2.9.2) has a quasi-inverse
$$\mathsf{funext} : \Big(\prod_{x:A} (f(x) = g(x))\Big) \to (f = g).$$

This function is also referred to as "function extensionality".

$$\begin{aligned}\mathsf{refl}_f &= \mathsf{funext}(x \mapsto \mathsf{refl}_{f(x)})\\ \alpha^{-1} &= \mathsf{funext}(x \mapsto \mathsf{happly}(\alpha, x)^{-1})\\ \alpha \cdot \beta &= \mathsf{funext}(x \mapsto \mathsf{happly}(\alpha, x) \cdot \mathsf{happly}(\beta, x)).\end{aligned}$$

Given a type $X$, a path $p : x_1 =_X x_2$, type families $A, B : X \to \mathcal{U}$, and a function $f : A(x_1) \to B(x_1)$, we have
$$\mathsf{transport}^{A\to B}(p, f) = \Big(x \mapsto \mathsf{transport}^B(p, f(\mathsf{transport}^A(p^{-1}, x)))\Big) \tag{2.9.4}$$
where $A \to B$ denotes abusively the type family $X \to \mathcal{U}$ defined by
$$(A \to B)(x) :\equiv (A(x) \to B(x)).$$

Transporting dependent functions is similar, but more complicated. Suppose given $X$ and $p$ as before, type families $A : X \to \mathcal{U}$ and $B : \prod_{(x:X)}(A(x) \to \mathcal{U})$, and also a dependent function $f : \prod_{(a:A(x_1))} B(x_1, a)$. Then for $a : A(x_2)$, we have
$$\mathsf{transport}^{\Pi_A(B)}(p, f)(a) =$$
$$\mathsf{transport}^{\widehat{B}}\Big((\mathsf{pair}^=(p^{-1}, \mathsf{refl}_{{p^{-1}}_*(a)}))^{-1}, f(\mathsf{transport}^A(p^{-1}, a))\Big)$$
where $\Pi_A(B)$ and $\widehat{B}$ denote respectively the type families
$$\begin{array}{rcll}\Pi_A(B) & :\equiv & \big(x \mapsto \prod_{(a:A(x))} B(x, a)\big) & : \quad X \to \mathcal{U}\\ \widehat{B} & :\equiv & \big(w \mapsto B(\mathsf{pr}_1 w, \mathsf{pr}_2 w)\big) & : \quad \big(\sum_{(x:X)} A(x)\big) \to \mathcal{U}.\end{array} \tag{2.9.5}$$

**Lemma 2.9.6.** *Given type families $A, B : X \to \mathcal{U}$ and $p : x =_X y$, and also $f : A(x) \to B(x)$ and $g : A(y) \to B(y)$, 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 $\widehat{q}$, then for $a : A(x)$, the path*
$$\mathsf{happly}(q, p_*(a)) : (p_*(f))(p_*(a)) = g(p_*(a))$$
*is equal to the concatenated path $i \cdot j \cdot k$, where*

- *$i : (p_*(f))(p_*(a)) = p_*\big(f({p^{-1}}_*(p_*(a)))\big)$ comes from (2.9.4),*
- *$j : p_*\big(f({p^{-1}}_*(p_*(a)))\big) = p_*(f(a))$ comes from Lemmas 2.1.3 and 2.3.9, and*
- *$k : p_*(f(a)) = g(p_*(a))$ is $\widehat{q}(a)$.*

**Lemma 2.9.7.** *Given type families $A : X \to \mathcal{U}$ and $B : \prod_{(x:X)} A(x) \to \mathcal{U}$ 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*
$$(p_*(f) = g) \simeq \Big(\prod_{a:A(x)} \mathsf{transport}^{\widehat{B}}(\mathsf{pair}^=(p, \mathsf{refl}_{p_*(a)}), f(a)) = g(p_*(a))\Big)$$
*with $\widehat{B}$ as in (2.9.5).*

## 2.10 Universes and the univalence axiom

**Lemma 2.10.1.** *For types $A, B : \mathcal{U}$, there is a certain function,*

$$\mathsf{idtoeqv} : (A =_{\mathcal{U}} B) \to (A \simeq B), \tag{2.10.2}$$

*defined in the proof.*

**Axiom 2.10.3** (Univalence)**.** *For any $A, B : \mathcal{U}$, the function (2.10.2) is an equivalence.*

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

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

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

**Lemma 2.10.4.** *For any type family $B : A \to \mathcal{U}$ and $x, y : A$ with a path $p : x = y$ and $u : B(x)$, we have*

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

## 2.11 Identity type

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

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

**Lemma 2.11.2.** *For any $A$ and $a : A$, with $p : x_1 = x_2$, we have*

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

**Theorem 2.11.3.** *For $f, g : A \to B$, with $p : a =_A a'$ and $q : f(a) =_B g(a)$, we have*

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

**Theorem 2.11.4.** *Let $B : A \to \mathcal{U}$ and $f, g : \prod_{(x:A)} B(x)$, with $p : a =_A a'$ and $q : f(a) =_{B(a)} g(a)$. Then we have*

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

**Theorem 2.11.5.** *For $p : a =_A a'$ with $q : a = a$ and $r : a' = a'$, we have*

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

## 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.

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

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

**Definition 2.14.1.** Given a type $A$, the type SemigroupStr(A) of **semigroup structures** with carrier $A$ is defined by

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

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

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

Moreover, this map is an equivalence, because $\mathsf{transport}^C(\alpha)$ is always an equivalence with inverse $\mathsf{transport}^C(\alpha^{-1})$, see Lemmas 2.1.3 and 2.3.9.

## 2.15 Universal properties

$$(X \to A \times B) \to (X \to A) \times (X \to B) \tag{2.15.1}$$

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

**Theorem 2.15.2.** *(2.15.1) is an equivalence.*

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

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

**Theorem 2.15.4.** *(2.15.3) is an equivalence.*

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

**Theorem 2.15.6.** *(2.15.5) is an equivalence.*

For pullbacks, the expected explicit construction works: given $f : A \to C$ and $g : B \to C$, we define

$$A \times_C B :\equiv \sum_{(a:A)} \sum_{(b:B)} (f(a) = g(b)). \tag{2.15.7}$$

*Exercise* 2.9. Prove that coproducts have the expected universal property,

$$(A + B \to X) \simeq (A \to X) \times (B \to X).$$

*Exercise* 2.10. Prove that $\Sigma$-types are "associative", in that for any $A : \mathcal{U}$ and families $B : A \to \mathcal{U}$ and $C : (\sum_{(x:A)} B(x)) \to \mathcal{U}$, we have

$$\left(\sum_{(x:A)} \sum_{(y:B(x))} C((x, y))\right) \simeq \left(\sum_{p:\sum_{(x:A)} B(x)} C(p)\right).$$

*Exercise* 2.11. A (homotopy) **commutative square**

$$\begin{tikzcd} P \arrow[r, "h"] \arrow[d, "k"'] & A \arrow[d, "f"] \\ B \arrow[r, "g"'] & C \end{tikzcd}$$

consists of functions $f$, $g$, $h$, and $k$ as shown, together with a path $f \circ h = g \circ k$. Note that this is exactly an element of the pullback $(P \to A) \times_{P \to C} (P \to B)$ as defined in (2.15.7). A commutative square is called a (homotopy) **pullback square** if for any $X$, the induced map

$$(X \to P) \to (X \to A) \times_{(X \to C)} (X \to B)$$

is an equivalence.

# Homotopy Type Theory

Sets and logic

## 3.1 Sets and $n$-types

**Definition 3.1.1.** 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 $\mathsf{isSet}(A)$ is defined to be the type

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

*Example* 3.1.2. 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.

*Example* 3.1.3. 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 $\mathbf{0}$.

*Example* 3.1.4. 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.

*Example* 3.1.5. 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 $p = \mathsf{pair}^{=}(\mathsf{ap}_{\mathsf{pr}_1}(p), \mathsf{ap}_{\mathsf{pr}_2}(p))$ and $q = \mathsf{pair}^{=}(\mathsf{ap}_{\mathsf{pr}_1}(q), \mathsf{ap}_{\mathsf{pr}_2}(q))$. But $\mathsf{ap}_{\mathsf{pr}_1}(p) = \mathsf{ap}_{\mathsf{pr}_1}(q)$ since $A$ is a set, and $\mathsf{ap}_{\mathsf{pr}_2}(p) = \mathsf{ap}_{\mathsf{pr}_2}(q)$ since $B$ is a set; hence $p = q$.
Similarly, if $A$ is a set and $B : A \to \mathcal{U}$ is such that each $B(x)$ is a set, then $\sum_{(x:A)} B(x)$ is a set.

*Example* 3.1.6. If $A$ is *any* type and $B : A \to \mathcal{U}$ 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

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

But for any $x : A$, we have

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

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

**Definition 3.1.7.** 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 $r = s$.

**Lemma 3.1.8.** *If $A$ is a set (that is, $\mathsf{isSet}(A)$ is inhabited), then $A$ is a 1-type.*

## 3.2 Propositions as types?

*Remark* 3.2.1. (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.

*Remark* 3.2.2. The classical *law of double negation* and *law of excluded middle* are incompatible with the univalence axiom.

**Theorem 3.2.3.** *It is not the case that for all $A : \mathcal{U}$ we have $\neg(\neg A) \to A$.*

*Remark* 3.2.4. For any $A$, $\neg\neg\neg A \to \neg A$ for any $A$.

**Corollary 3.2.5.** *It is not the case that for all $A : \mathcal{U}$ we have $A + (\neg A)$.*

## 3.3 Mere propositions

**Definition 3.3.1.** 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 $\mathsf{isProp}(P)$ is defined to be

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

**Lemma 3.3.2.** *If $P$ is a mere proposition and $x_0 : P$, then $P \simeq \mathbf{1}$.*

**Lemma 3.3.3.** *If $P$ and $Q$ are mere propositions such that $P \to Q$ and $Q \to P$, then $P \simeq Q$.*

**Lemma 3.3.4.** *Every mere proposition is a set.*

**Lemma 3.3.5.** *For any type $A$, the types $\mathsf{isProp}(A)$ and $\mathsf{isSet}(A)$ are mere propositions.*

## 3.4 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:

$$\mathsf{LEM} :\equiv \prod_{A:\mathcal{U}} \Big(\mathsf{isProp}(A) \to (A + \neg A)\Big). \tag{3.4.1}$$

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

$$\prod_{A:\mathcal{U}} \Big(\mathsf{isProp}(A) \to (\neg\neg A \to A)\Big). \tag{3.4.2}$$

**Definition 3.4.3.**

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

## 3.5 Subsets and propositional resizing

**Lemma 3.5.1.** *Suppose $P : A \to \mathcal{U}$ 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 $\mathsf{pr}_1(u) = \mathsf{pr}_1(v)$, then $u = v$.*

For instance, recall that

$$(A \simeq B) :\equiv \sum_{f:A \to B} \mathsf{isequiv}(f),$$

where each type $\mathsf{isequiv}(f)$ was supposed to be a mere proposition. It follows that if two equivalences have equal underlying functions, then they are equal as equivalences.
If $P : A \to \mathcal{U}$ is a family of mere propositions (i.e. each $P(x)$ is a mere proposition), we may write

$$\{\, x : A \mid P(x) \,\} \tag{3.5.2}$$

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

$$\mathsf{Set}_{\mathcal{U}} :\equiv \{\, A : \mathcal{U} \mid \mathsf{isSet}(A) \,\},$$
$$\mathsf{Prop}_{\mathcal{U}} :\equiv \{\, A : \mathcal{U} \mid \mathsf{isProp}(A) \,\}.$$

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

**Axiom 3.5.3** (Propositional resizing)**.** *The map $\mathsf{Prop}_{\mathcal{U}_i} \to \mathsf{Prop}_{\mathcal{U}_{i+1}}$ is an equivalence.*

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

$$\mathcal{P}(A) :\equiv (A \to \Omega),$$

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

## 3.6 The logic of mere propositions

*Example* 3.6.1. 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 Example 3.1.5 but simpler. Thus, the connective "and" preserves mere propositions.

*Example* 3.6.2. If $A$ is any type and $B : A \to \mathcal{U}$ 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 Example 3.1.6 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 $A \to B$ regardless of what $A$ is. In even more particular, since $\mathbf{0}$ is a mere proposition, so is $\neg A \equiv (A \to \mathbf{0})$. Thus, the connectives "implies" and "not" preserve mere propositions, as does the quantifier "for all".

## 3.7 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 $f : A \to B$, then there is an induced $g : \|A\| \to B$ such that $g(|a|) \equiv f(a)$ for all $a : A$.

**Definition 3.7.1.** We define **traditional logical notation** using truncation as follows, where $P$ and $Q$ denote mere propositions (or families thereof):

$$\begin{aligned}
\top &:\equiv \mathbf{1}\\
\bot &:\equiv \mathbf{0}\\
P \wedge Q &:\equiv P \times Q\\
P \Rightarrow Q &:\equiv P \to Q\\
P \Leftrightarrow Q &:\equiv P = Q\\
\neg P &:\equiv P \to \mathbf{0}\\
P \vee Q &:\equiv \|P + Q\|\\
\forall (x : A).\, P(x) &:\equiv \prod_{x:A} P(x)\\
\exists (x : A).\, P(x) &:\equiv \Big\| \sum_{x:A} P(x) \Big\|
\end{aligned}$$

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

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

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 $A$.

## 3.8 The axiom of choice

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

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,

$$\Big(\prod_{x:X} \Big\| \sum_{a:A(x)} P(x,a) \Big\|\Big) \to \Big\| \sum_{(g:\prod_{(x:X)} A(x))} \prod_{(x:X)} P(x, g(x)) \Big\|. \tag{3.8.1}$$

Of course, this is a direct translation of (3.2.1) 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

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

**Lemma 3.8.2.** *The axiom of choice (3.8.1) is equivalent to the statement that for any set $X$ and any $Y : X \to \mathcal{U}$ such that each $Y(x)$ is a set, we have*

$$\Big(\prod_{x:X} \big\|Y(x)\big\|\Big) \to \Big\|\prod_{x:X} Y(x)\Big\|. \tag{3.8.3}$$

*Remark* 3.8.4. The right side of (3.8.3) always implies the left. Since both are mere propositions, by Lemma 3.3.3 the axiom of choice is also equivalent to asking for an equivalence

$$\Big(\prod_{x:X} \big\|Y(x)\big\|\Big) \simeq \Big\|\prod_{x:X} Y(x)\Big\|$$

**Lemma 3.8.5.** *There exists a type $X$ and a family $Y : X \to \mathcal{U}$ such that each $Y(x)$ is a set, but such that (3.8.3) is false.*

## 3.9 The principle of unique choice

**Lemma 3.9.1.** *If $P$ is a mere proposition, then $P \simeq \|P\|$.*

**Corollary 3.9.2** (The principle of unique choice)**.** *Suppose a type family $P : A \to \mathcal{U}$ such that*

*(i). For each $x$, the type $P(x)$ is a mere proposition, and*
*(ii). For each $x$ we have $\|P(x)\|$.*

*Then we have $\prod_{(x:A)} P(x)$.*

## 3.11 Contractibility

**Definition 3.11.1.** 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 $\mathsf{contr}_x$.

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

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

**Lemma 3.11.2.** *For a type $A$, the following are logically equivalent.*

*(i). $A$ is contractible in the sense of Definition 3.11.1.*
*(ii). $A$ is a mere proposition, and there is a point $a : A$.*
*(iii). $A$ is equivalent to $\mathbf{1}$.*

**Lemma 3.11.3.** *For any type $A$, the type $\mathsf{isContr}(A)$ is a mere proposition.*

**Corollary 3.11.4.** *If $A$ is contractible, then so is $\mathsf{isContr}(A)$.*

**Lemma 3.11.5.** *If $P : A \to \mathcal{U}$ 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 $r : A \to B$ such that there exists a function $s : B \to A$, called its **section**, and a homotopy $\epsilon : \prod_{(y:B)} (r(s(y)) = y)$; then we say that $B$ is a **retract**of $A$.

**Lemma 3.11.6.** *If $B$ is a retract of $A$, and $A$ is contractible, then so is $B$.*

**Lemma 3.11.7.** *For any $A$ and any $a : A$, the type $\sum_{(x:A)} (a = x)$ is contractible.*

**Lemma 3.11.8.** *Let $P : A \to \mathcal{U}$ be a type family.*

*(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 $P(a)$.*

**Lemma 3.11.9.** *A type $A$ is a mere proposition if and only if for all $x, y : A$, the type $x =_A y$ is contractible.*

# Homotopy Type Theory

## Equivalences

Recall that we wanted $\mathsf{isequiv}(f)$ to have the following properties, which we restate here:

(i). $\mathsf{qinv}(f) \to \mathsf{isequiv}(f)$.
(ii). $\mathsf{isequiv}(f) \to \mathsf{qinv}(f)$.
(iii). $\mathsf{isequiv}(f)$ is a mere proposition.

Here $\mathsf{qinv}(f)$ denotes the type of quasi-inverses to $f$:

$$\sum_{g:B\to A} \big((f\circ g \sim \mathsf{id}_B)\times(g\circ f\sim \mathsf{id}_A)\big).$$

By function extensionality, it follows that $\mathsf{qinv}(f)$ is equivalent to the type

$$\sum_{g:B\to A} \big((f\circ g = \mathsf{id}_B)\times(g\circ f= \mathsf{id}_A)\big).$$

We will define three different types having properties (i)–(iii), which we call

- half adjoint equivalences,
- bi-invertible maps, and
- contractible functions.

## 4.1 Quasi-inverses

**Lemma 4.1.1.** *If $f : A \to B$ is such that $\mathsf{qinv}(f)$ is inhabited, then*

$$\mathsf{qinv}(f) \simeq \Big(\prod_{x:A}(x=x)\Big).$$

**Lemma 4.1.2.** *Suppose we have a type $A$ with $a : A$ and $q : a = a$ such that*

*(i). The type $a = a$ is a set.*
*(ii). For all $x : A$ we have $\|a = x\|$.*
*(iii). For all $p : a = a$ we have $p \cdot q = q \cdot p$.*

*Then there exists $f : \prod_{(x:A)}(x = x)$ with $f(a) = q$.*

**Theorem 4.1.3.** *There exist types $A$ and $B$ and a function $f : A \to B$ such that $\mathsf{qinv}(f)$ is not a mere proposition.*

## 4.2 Half adjoint equivalences

**Definition 4.2.1.** A function $f : A \to B$ is a **half adjoint equivalence** if there are $g : B \to A$ and homotopies $\eta : g\circ f \sim \mathsf{id}_A$ and $\epsilon : f \circ g \sim \mathsf{id}_B$ such that there exists a homotopy

$$\tau : \prod_{x:A} f(\eta x) = \epsilon(fx).$$

**Lemma 4.2.2.** *For functions $f : A \to B$ and $g : B \to A$ and homotopies $\eta : g \circ f \sim \mathsf{id}_A$ and $\epsilon : f \circ g \sim \mathsf{id}_B$, the following conditions are logically equivalent:*

- $\prod_{(x:A)} f(\eta x) = \epsilon(fx)$
- $\prod_{(y:B)} g(\epsilon y) = \eta(gy)$

**Theorem 4.2.3.** *For any $f : A \to B$ we have $\mathsf{qinv}(f) \to \mathsf{ishae}(f)$.*

**Definition 4.2.4.** The **fiber** of a map $f : A \to B$ over a point $y : B$ is

$$\mathsf{fib}_f(y) :\equiv \sum_{x:A}(f(x) = y).$$

In homotopy theory, this is what would be called the *homotopy fiber* of $f$. The path lemmas in §2.5 yield the following characterization of paths in fibers:

**Lemma 4.2.5.** *For any $f : A \to B$, $y : B$, and $(x,p),(x',p') : \mathsf{fib}_f(y)$, we have*

$$((x,p) = (x',p')) \simeq \Big(\sum_{\gamma:x=x'} f(\gamma)\cdot p' = p\Big)$$

**Theorem 4.2.6.** *If $f : A \to B$ is a half adjoint equivalence, then for any $y : B$ the fiber $\mathsf{fib}_f(y)$ is contractible.*

**Definition 4.2.7.** Given a function $f : A \to B$, we define the types

$$\mathsf{linv}(f) :\equiv \sum_{g:B\to A}(g\circ f \sim \mathsf{id}_A)$$
$$\mathsf{rinv}(f) :\equiv \sum_{g:B\to A}(f\circ g \sim \mathsf{id}_B)$$

of **left inverses** and **right inverses** to $f$, respectively. We call $f$ **left invertible** if $\mathsf{linv}(f)$ is inhabited, and similarly **right invertible** if $\mathsf{rinv}(f)$ is inhabited.

**Lemma 4.2.8.** *If $f : A \to B$ has a quasi-inverse, then so do*

$$(f\circ -) : (C\to A) \to (C \to B)$$
$$(-\circ f) : (B\to C) \to (A \to C).$$

**Lemma 4.2.9.** *If $f : A \to B$ has a quasi-inverse, then the types $\mathsf{rinv}(f)$ and $\mathsf{linv}(f)$ are contractible.*

**Definition 4.2.10.** For $f : A \to B$, a left inverse $(g, \eta) : \mathsf{linv}(f)$, and a right inverse $(g,\epsilon) : \mathsf{rinv}(f)$, we denote

$$\mathsf{lcoh}_f(g,\eta) :\equiv \sum_{(\epsilon:f\circ g\sim\mathsf{id}_B)}\ \prod_{(y:B)} g(\epsilon y) = \eta(gy),$$
$$\mathsf{rcoh}_f(g,\epsilon) :\equiv \sum_{(\eta:g\circ f\sim\mathsf{id}_A)}\ \prod_{(x:A)} f(\eta x) = \epsilon(fx).$$

**Lemma 4.2.11.** *For any $f, g, \epsilon, \eta$, we have*

$$\mathsf{lcoh}_f(g,\eta) \simeq \prod_{y:B}(fgy, \eta(gy)) =_{\mathsf{fib}_g(gy)} (y, \mathsf{refl}_{gy}),$$
$$\mathsf{rcoh}_f(g,\epsilon) \simeq \prod_{x:A}(gfx, \epsilon(fx)) =_{\mathsf{fib}_f(fx)} (x, \mathsf{refl}_{fx}).$$

**Lemma 4.2.12.** *If $f$ is a half adjoint equivalence, then for any $(g,\epsilon) : \mathsf{rinv}(f)$, the type $\mathsf{rcoh}_f(g,\epsilon)$ is contractible.*

**Theorem 4.2.13.** *For any $f : A \to B$, the type $\mathsf{ishae}(f)$ is a mere proposition.*

## 4.3 Bi-invertible maps

**Definition 4.3.1.** We say $f : A \to B$ is **bi-invertible** if it has both a left inverse and a right inverse:

$$\mathsf{biinv}(f) :\equiv \mathsf{linv}(f)\times\mathsf{rinv}(f).$$

**Theorem 4.3.2.** *For any $f : A \to B$, the type $\mathsf{biinv}(f)$ is a mere proposition.*

**Corollary 4.3.3.** *For any $f : A \to B$ we have $\mathsf{biinv}(f) \simeq \mathsf{ishae}(f)$.*

## 4.4 Contractible fibers

**Definition 4.4.1** (Contractible maps)**.** A map $f : A \to B$ is **contractible** if for all $y : B$, the fiber $\mathsf{fib}_f(y)$ is contractible.

**Theorem 4.4.2.** *For any $f : A \to B$ we have $\mathsf{isContr}(f) \to \mathsf{ishae}(f)$.*

**Lemma 4.4.3.** *For any $f$, the type $\mathsf{isContr}(f)$ is a mere proposition.*

**Theorem 4.4.4.** *For any $f : A \to B$ we have $\mathsf{isContr}(f) \simeq \mathsf{ishae}(f)$.*

**Corollary 4.4.5.** *If $f : A \to B$ is such that $B \to \mathsf{isequiv}(f)$, then $f$ is an equivalence.*

## 4.5 On the definition of equivalences

We have shown that all three definitions of equivalence satisfy the three desirable properties and are pairwise equivalent:

$$\mathsf{isContr}(f) \simeq \mathsf{ishae}(f) \simeq \mathsf{biinv}(f).$$

## 4.6 Surjections and embeddings

When $A$ and $B$ are sets and $f : A \to B$ is an equivalence, we also call it as **isomorphism** or a **bijection**. (We avoid these words for types that are not sets, since in homotopy theory and higher category theory they often denote a stricter notion of "sameness" than homotopy equivalence.) In set theory, a function is a bijection just when it is both injective and surjective. The same is true in type theory, if we formulate these conditions appropriately. For clarity, when dealing with types that are not sets, we will speak of *embeddings* instead of injections.

**Definition 4.6.1.** Let $f : A \to B$.

(i). We say $f$ is **surjective** (or a **surjection**) if for every $b : B$ we have $\|\mathsf{fib}_f(b)\|$.
(ii). We say $f$ is an **embedding** if for every $x, y : A$ the function $\mathsf{ap}_f : (x =_A y) \to (f(x) =_B f(y))$ is an equivalence.

In other words, $f$ is surjective if every fiber of $f$ is merely inhabited, or equivalently if for all $b : B$ there merely exists an $a : A$ such that $f(a) = b$. In traditional logical notation, $f$ is surjective if $\forall(b : B). \exists(a : A). (f(a) = b)$. This must be distinguished from the stronger assertion that $\prod_{(b:B)} \sum_{(a:A)} (f(a) = b)$; if this holds we say that $f$ is a **split surjection**. (Since this latter type is equivalent to $\sum_{(g:B\to A)} \prod_{(b:B)} (f(g(b)) = b)$, being a split surjection is the same as being a *retraction* as defined in §3.11.)
The axiom of choice from §3.8 says exactly that every surjection *between sets* is split. However, in the presence of the univalence axiom, it is simply false that *all* surjections are split. In Lemma 3.8.5 we constructed a type family $Y : X \to \mathcal{U}$ such that $\prod_{(x:X)} \|Y(x)\|$ but $\neg \prod_{(x:X)} Y(x)$; for any such family, the first projection $(\sum_{(x:X)} Y(x)) \to X$ is a surjection that is not split.
If $A$ and $B$ are sets, then by Lemma 3.3.3, $f$ is an embedding just when

$$\prod_{x,y:A} (f(x) =_B f(y)) \to (x =_A y). \tag{4.6.2}$$

In this case we say that $f$ is **injective**, or an **injection**. We avoid these word for types that are not sets, because they might be interpreted as (4.6.2), which is an ill-behaved notion for non-sets. It is also true that any function between sets is surjective if and only if it is an *epimorphism* in a suitable sense, but this also fails for more general types, and surjectivity is generally the more important notion.

**Theorem 4.6.3.** *A function $f : A \to B$ is an equivalence if and only if it is both surjective and an embedding.*

**Corollary 4.6.4.** *For any $f : A \to B$ we have*

$$\mathsf{isequiv}(f) \simeq (\mathsf{isEmbedding}(f) \times \mathsf{isSurjective}(f)).$$

## 4.7 Closure properties of equivalences

**Theorem 4.7.1** (The 2-out-of-3 property). *Suppose $f : A \to B$ and $g : B \to C$. If any two of $f$, $g$, and $g \circ f$ are equivalences, so is the third.*

**Definition 4.7.2.** A function $g : A \to B$ is said to be a **retract** of a function $f : X \to Y$ if there is a diagram

$$\begin{tikzcd}
A \arrow[r, "s"] \arrow[d, "g"'] & X \arrow[r, "r"] \arrow[d, "f"] & A \arrow[d, "g"] \\
B \arrow[r, "s'"'] & Y \arrow[r, "r'"'] & B
\end{tikzcd}$$

for which there are

(i). a homotopy $R : r \circ s \sim \mathsf{id}_A$.
(ii). a homotopy $R' : r' \circ s' \sim \mathsf{id}_B$.
(iii). a homotopy $L : f \circ s \sim s' \circ g$.
(iv). a homotopy $K : g \circ r \sim r' \circ f$.
(v). for every $a : A$, a path $H(a)$ witnessing the commutativity of the square

$$\begin{tikzcd}
g(r(s(a))) \arrow[r, equal, "K(s(a))"] \arrow[d, equal, "g(R(a))"'] & r'(f(s(a))) \arrow[d, equal, "r'(L(a))"] \\
g(a) \arrow[r, equal, "R'(g(a))^{-1}"'] & r'(s'(g(a)))
\end{tikzcd}$$

**Lemma 4.7.3.** *If a function $g : A \to B$ is a retract of a function $f : X \to Y$, then $\mathsf{fib}_g(b)$ is a retract of $\mathsf{fib}_f(s'(b))$ for every $b : B$, where $s' : B \to Y$ is as in Definition 4.7.2.*

**Theorem 4.7.4.** *If $g$ is a retract of an equivalence $f$, then $g$ is also an equivalence.*

**Definition 4.7.5.** Given type families $P, Q : A \to \mathcal{U}$ and a map $f : \prod_{(x:A)} P(x) \to Q(x)$, we define

$$\mathsf{total}(f) :\equiv \lambda w. (\mathsf{pr}_1 w, f(\mathsf{pr}_1 w, \mathsf{pr}_2 w)) : \sum_{x:A} P(x) \to \sum_{x:A} Q(x).$$

**Theorem 4.7.6.** *Suppose that $f$ is a fiberwise transformation between families $P$ and $Q$ over a type $A$ and let $x : A$ and $v : Q(x)$. Then we have an equivalence*

$$\mathsf{fib}_{\mathsf{total}(f)}((x, v)) \simeq \mathsf{fib}_{f(x)}(v).$$

**Theorem 4.7.7.** *Suppose that $f$ is a fiberwise transformation between families $P$ and $Q$ over a type $A$. Then $f$ is a fiberwise equivalence if and only if $\mathsf{total}(f)$ is an equivalence.*

## 4.8 The object classifier

**Lemma 4.8.1.** *For any type family $B : A \to \mathcal{U}$, the fiber of $\mathsf{pr}_1 : \sum_{(x:A)} B(x) \to A$ over $a : A$ is equivalent to $B(a)$:*

$$\mathsf{fib}_{\mathsf{pr}_1}(a) \simeq B(a)$$

**Lemma 4.8.2.** *For any function $f : A \to B$, we have $A \simeq \sum_{(b:B)} \mathsf{fib}_f(b)$.*

**Theorem 4.8.3.** *For any type $B$ there is an equivalence*

$$\chi : \Big(\sum_{A:\mathcal{U}} (A \to B)\Big) \simeq (B \to \mathcal{U}).$$

**Theorem 4.8.4.** *Let $f : A \to B$ be a function. Then the diagram*

$$\begin{tikzcd}
A \arrow[r, "\vartheta_f"] \arrow[d, "f"'] & \mathcal{U}_\bullet \arrow[d, "\mathsf{pr}_1"] \\
B \arrow[r, "\chi_f"'] & \mathcal{U}
\end{tikzcd}$$

*is a pullback square. (see Exercise 2.11). Here the function $\vartheta_f$ is defined by*

$$\lambda a. (\mathsf{fib}_f(f(a)), (a, \mathsf{refl}_{f(a)})).$$

## 4.9 Univalence implies function extensionality

**Definition 4.9.1.** The **weak function extensionality principle** asserts that there is a function

$$\Big(\prod_{x:A} \mathsf{isContr}(P(x))\Big) \to \mathsf{isContr}\Big(\prod_{x:A} P(x)\Big)$$

for any family $P : A \to \mathcal{U}$ of types over any type $A$.

**Lemma 4.9.2.** *Assuming $\mathcal{U}$ is univalent, for any $A, B, X : \mathcal{U}$ and any $e : A \simeq B$, there is an equivalence*

$$(X \to A) \simeq (X \to B)$$

*of which the underlying map is given by post-composition with the underlying function of $e$.*

**Corollary 4.9.3.** *Let $P : A \to \mathcal{U}$ be a family of contractible types, i.e. $\prod_{(x:A)} \mathsf{isContr}(P(x))$. Then the projection $\mathsf{pr}_1 : (\sum_{(x:A)} P(x)) \to A$ is an equivalence. Assuming $\mathcal{U}$ is univalent, it follows immediately that post-composition with $\mathsf{pr}_1$ gives an equivalence*

$$\alpha : \Big(A \to \sum_{x:A} P(x)\Big) \simeq (A \to A).$$

**Theorem 4.9.4.** *In a univalent universe $\mathcal{U}$, suppose that $P : A \to \mathcal{U}$ is a family of contractible types and let $\alpha$ be the function of Corollary 4.9.3. Then $\prod_{(x:A)} P(x)$ is a retract of $\mathsf{fib}_\alpha(\mathsf{id}_A)$. As a consequence, $\prod_{(x:A)} P(x)$ is contractible. In other words, the univalence axiom implies the weak function extensionality principle.*

**Theorem 4.9.5.** *Weak function extensionality implies the function extensionality Axiom 2.9.3.*

# Homotopy Type Theory

## Categories

## 9.1 Categories and precategories

**Definition 9.1.1.** A **precategory** $A$ consists of the following.

(i). A type $A_0$, whose elements are called **objects**. We write $a : A$ for $a : A_0$.

(ii). For each $a, b : A$, a set $\hom_A(a, b)$, whose elements are called **arrows** or **morphisms**.

(iii). For each $a : A$, a morphism $1_a : \hom_A(a, a)$, called the **identity morphism**.

(iv). For each $a, b, c : A$, a function

$$\hom_A(b, c) \to \hom_A(a, b) \to \hom_A(a, c)$$

called **composition**, and denoted infix by $g \mapsto f \mapsto g \circ f$, or sometimes simply by $gf$.

(v). For each $a, b : A$ and $f : \hom_A(a, b)$, we have $f = 1_b \circ f$ and $f = f \circ 1_a$.

(vi). For each $a, b, c, d : A$ and

$$f : \hom_A(a, b), \qquad g : \hom_A(b, c), \qquad h : \hom_A(c, d),$$

we have $h \circ (g \circ f) = (h \circ g) \circ f$.

**Definition 9.1.2.** A morphism $f : \hom_A(a, b)$ is an **isomorphism** if there is a morphism $g : \hom_A(b, a)$ such that $g \circ f = 1_a$ and $f \circ g = 1_b$. We write $a \cong b$ for the type of such isomorphisms.

**Lemma 9.1.3.** *For any $f : \hom_A(a, b)$, the type "$f$ is an isomorphism" is a mere proposition. Therefore, for any $a, b : A$ the type $a \cong b$ is a set.*

**Lemma 9.1.4** (idtoiso). *If $A$ is a precategory and $a, b : A$, then*

$$(a = b) \to (a \cong b).$$

*Example* 9.1.5. There is a precategory $\mathcal{S}et$, whose type of objects is Set, and with $\hom_{\mathcal{S}et}(A, B) :\equiv (A \to B)$. The identity morphisms are identity functions and the composition is function composition. For this precategory, Lemma 9.1.4 is equal to (the restriction to sets of) the map idtoeqv from §2.10.
Of course, to be more precise we should call this category $\mathcal{S}et_{\mathcal{U}}$, since its objects are only the *small sets* relative to a universe $\mathcal{U}$.

**Definition 9.1.6.** A **category** is a precategory such that for all $a, b : A$, the function $\mathsf{idtoiso}_{a,b}$ from Lemma 9.1.4 is an equivalence.

*Example* 9.1.7. The univalence axiom implies immediately that $\mathcal{S}et$ is a category. One can also show, using univalence, that any precategory of set-level structures such as groups, rings, topological spaces, etc. is a category; see §9.8.

**Lemma 9.1.8.** *In a category, the type of objects is a 1-type.*

**Lemma 9.1.9.** *For $p : a = a'$ and $q : b = b'$ and $f : \hom_A(a, b)$, we have*

$$(p, q)_*(f) = \mathsf{idtoiso}(q) \circ f \circ \mathsf{idtoiso}(p)^{-1}. \qquad (9.1.10)$$

*Example* 9.1.11. A precategory in which each set $\hom_A(a, b)$ is a mere proposition is equivalently a type $A_0$ equipped with a mere relation "$\leq$" that is reflexive ($a \leq a$) and transitive (if $a \leq b$ and $b \leq c$, then $a \leq c$). We call this a **preorder**.
In a preorder, a witness $f : a \leq b$ is an isomorphism just when there exists some witness $g : b \leq a$. Thus, $a \cong b$ is the mere proposition that $a \leq b$ and $b \leq a$. Therefore, a preorder $A$ is a category just when (1) each type $a = b$ is a mere proposition, and (2) for any $a, b : A_0$ there exists a function $(a \cong b) \to (a = b)$. In other words, $A_0$ must be a set, and $\leq$ must be antisymmetric (if $a \leq b$ and $b \leq a$, then $a = b$). We call this a **(partial) order** or a **poset**.

*Example* 9.1.12. If $A$ is a category, then $A_0$ is a set if and only if for any $a, b : A_0$, the type $a \cong b$ is a mere proposition. This is equivalent to saying that every isomorphism in $A$ is an identity; thus it is rather stronger than the classical notion of "skeletal" category. Categories of this sort are sometimes called **gaunt** [?]. There is not really any notion of "skeletality" for our categories, unless one considers Definition 9.1.6 itself to be such.

*Example* 9.1.13. For any 1-type $X$, there is a category with $X$ as its type of objects and with $\hom(x, y) :\equiv (x = y)$. If $X$ is a set, we call this the **discrete** category on $X$. In general, we call it a **groupoid** (see **??**).

*Example* 9.1.14. For *any* type $X$, there is a precategory with $X$ as its type of objects and with $\hom(x, y) :\equiv \|x = y\|_0$. The composition operation

$$\|y = z\|_0 \to \|x = y\|_0 \to \|x = z\|_0$$

is defined by induction on truncation from concatenation $(y = z) \to (x = y) \to (x = z)$. We call this the **fundamental pregroupoid** of $X$. (In fact, we have met it already in **??**; see also **??**.)

*Example* 9.1.15. There is a precategory whose type of objects is $\mathcal{U}$ and with $\hom(X, Y) :\equiv \|X \to Y\|_0$, and composition defined by induction on truncation from ordinary composition $(Y \to Z) \to (X \to Y) \to (X \to Z)$. We call this the **homotopy precategory of types**.

*Example* 9.1.16. Let $\mathcal{R}el$ be the following precategory:

- Its objects are sets.
- $\hom_{\mathcal{R}el}(X, Y) = X \to Y \to \mathsf{Prop}$.
- For a set $X$, we have $1_X(x, x') :\equiv (x = x')$.
- For $R : \hom_{\mathcal{R}el}(X, Y)$ and $S : \hom_{\mathcal{R}el}(Y, Z)$, their composite is defined by

$$(S \circ R)(x, z) :\equiv \left\| \sum_{y:Y} R(x, y) \times S(y, z) \right\|.$$

Suppose $R : \hom_{\mathcal{R}el}(X, Y)$ is an isomorphism, with inverse $S$. We observe the following.

(i). If $R(x, y)$ and $S(y', x)$, then $(R \circ S)(y', y)$, and hence $y' = y$. Similarly, if $R(x, y)$ and $S(y, x')$, then $x = x'$.

(ii). For any $x$, we have $x = x$, hence $(S \circ R)(x, x)$. Thus, there merely exists a $y : Y$ such that $R(x, y)$ and $S(y, x)$.

(iii). Suppose $R(x, y)$. By (ii), there merely exists a $y'$ with $R(x, y')$ and $S(y', x)$. But then by (i), merely $y' = y$, and hence $y' = y$ since $Y$ is a set. Therefore, by transporting $S(y', x)$ along this equality, we have $S(y, x)$. In conclusion, $R(x, y) \to S(y, x)$. Similarly, $S(y, x) \to R(x, y)$.

(iv). If $R(x, y)$ and $R(x, y')$, then by (iii), $S(y', x)$, so that by (i), $y = y'$. Thus, for any $x$ there is at most one $y$ such that $R(x, y)$. And by (ii), there merely exists such a $y$, hence there exists such a $y$.

In conclusion, if $R : \hom_{\mathcal{R}el}(X, Y)$ is an isomorphism, then for each $x : X$ there is exactly one $y : Y$ such that $R(x, y)$, and dually. Thus, there is a function $f : X \to Y$ sending each $x$ to this $y$, which is an equivalence; hence $X = Y$. With a little more work, we conclude that $\mathcal{R}el$ is a category.

## 9.2 Functors and transformations

**Definition 9.2.1.** Let $A$ and $B$ be precategories. A **functor** $F : A \to B$ consists of

(i). A function $F_0 : A_0 \to B_0$, generally also denoted $F$.

(ii). For each $a, b : A$, a function $F_{a,b} : \hom_A(a, b) \to \hom_B(Fa, Fb)$, generally also denoted $F$.

(iii). For each $a : A$, we have $F(1_a) = 1_{Fa}$.

(iv). For each $a, b, c : A$ and $f : \hom_A(a, b)$ and $g : \hom_A(b, c)$, we have

$$F(g \circ f) = Fg \circ Ff.$$

**Definition 9.2.2.** For functors $F, G : A \to B$, a **natural transformation** $\gamma : F \to G$ consists of

(i). For each $a : A$, a morphism $\gamma_a : \hom_B(Fa, Ga)$ (the "components").

(ii). For each $a, b : A$ and $f : \hom_A(a, b)$, we have $Gf \circ \gamma_a = \gamma_b \circ Ff$ (the "naturality axiom").

**Definition 9.2.3.** For precategories $A, B$, there is a precategory $B^A$, called the **functor precategory**, defined by

- $(B^A)_0$ is the type of functors from $A$ to $B$.
- $\hom_{B^A}(F, G)$ is the type of natural transformations from $F$ to $G$.

**Lemma 9.2.4.** *A natural transformation $\gamma : F \to G$ is an isomorphism in $B^A$ if and only if each $\gamma_a$ is an isomorphism in $B$.*

**Theorem 9.2.5.** *If $A$ is a precategory and $B$ is a category, then $B^A$ is a category.*

**Definition 9.2.6.** For functors $F : A \to B$ and $G : B \to C$, their composite $G \circ F : A \to C$ is given by

- The composite $(G_0 \circ F_0) : A_0 \to C_0$
- For each $a, b : A$, the composite

$$(G_{Fa,Fb} \circ F_{a,b}) : \hom_A(a, b) \to \hom_C(GFa, GFb).$$

It is easy to check the axioms.

**Definition 9.2.7.** For functors $F : A \to B$ and $G, H : B \to C$ and a natural transformation $\gamma : G \to H$, the composite $(\gamma F) : GF \to HF$ is given by

- For each $a : A$, the component $\gamma_{Fa}$.

Naturality is easy to check. Similarly, for $\gamma$ as above and $K : C \to D$, the composite $(K\gamma) : KG \to KH$ is given by

- For each $b : B$, the component $K(\gamma_b)$.

**Lemma 9.2.8.** *For functors $F, G : A \to B$ and $H, K : B \to C$ and natural transformations $\gamma : F \to G$ and $\delta : H \to K$, we have*

$$(\delta G)(H\gamma) = (K\gamma)(\delta F).$$

**Lemma 9.2.9.** *Composition of functors is associative: $H(GF) = (HG)F$.*

**Lemma 9.2.10.** *Lemma 9.2.9 is coherent, i.e. the following pentagon of equalities commutes:*

$$\begin{tikzcd}
 & K(H(GF)) \arrow[dl, equal] \arrow[dr, equal] & \\
(KH)(GF) \arrow[d, equal] & & K((HG)F) \arrow[d, equal] \\
((KH)G)F \arrow[rr, equal] & & (K(HG))F
\end{tikzcd}$$

**Lemma 9.2.11.** *For a functor $F : A \to B$, we have equalities $(1_B \circ F) = F$ and $(F \circ 1_A) = F$, such that given also $G : B \to C$, the following triangle of equalities commutes.*

$$\begin{tikzcd}
G \circ (1_B \circ F) \arrow[rr, equal] \arrow[dr, equal] & & (G \circ 1_B) \circ F \arrow[dl, equal] \\
 & G \circ F. &
\end{tikzcd}$$

## 9.3 Adjunctions

**Definition 9.3.1.** A functor $F : A \to B$ is a **left adjoint** if there exists

- A functor $G : B \to A$.
- A natural transformation $\eta : 1_A \to GF$ (the **unit**).
- A natural transformation $\epsilon : FG \to 1_B$ (the **counit**).
- $(\epsilon F)(F\eta) = 1_F$.
- $(G\epsilon)(\eta G) = 1_G$.

**Lemma 9.3.2.** *If $A$ is a category (but $B$ may be only a precategory), then the type "$F$ is a left adjoint" is a mere proposition.*

## 9.4 Equivalences

**Definition 9.4.1.** A functor $F : A \to B$ is an **equivalence of (pre)categories** if it is a left adjoint for which $\eta$ and $\epsilon$ are isomorphisms. We write $A \simeq B$ for the type of equivalences of categories from $A$ to $B$.

**Lemma 9.4.2.** *If for $F : A \to B$ there exists $G : B \to A$ and isomorphisms $GF \cong 1_A$ and $FG \cong 1_B$, then $F$ is an equivalence of precategories.*

**Definition 9.4.3.** We say a functor $F : A \to B$ is **faithful** if for all $a, b : A$, the function

$$F_{a,b} : \hom_A(a, b) \to \hom_B(Fa, Fb)$$

is injective, and **full** if for all $a, b : A$ this function is surjective. If it is both (hence each $F_{a,b}$ is an equivalence) we say $F$ is **fully faithful**.

**Definition 9.4.4.** We say a functor $F : A \to B$ is **split essentially surjective** if for all $b : B$ there exists an $a : A$ such that $Fa \cong b$.

**Lemma 9.4.5.** *For any precategories $A$ and $B$ and functor $F : A \to B$, the following types are equivalent.*

(i). *$F$ is an equivalence of precategories.*
(ii). *$F$ is fully faithful and split essentially surjective.*

**Definition 9.4.6.** A functor $F : A \to B$ is **essentially surjective** if for all $b : B$, there *merely* exists an $a : A$ such that $Fa \cong b$. We say $F$ is a **weak equivalence** if it is fully faithful and essentially surjective.

**Lemma 9.4.7.** *If $F : A \to B$ is fully faithful and $A$ is a category, then for any $b : B$ the type $\sum_{(a:A)}(Fa \cong b)$ is a mere proposition. Hence a functor between categories is an equivalence if and only if it is a weak equivalence.*

**Definition 9.4.8.** A functor $F : A \to B$ is an **isomorphism of (pre)categories** if $F$ is fully faithful and $F_0 : A_0 \to B_0$ is an equivalence of types.

**Lemma 9.4.9.** *For precategories $A$ and $B$ and $F : A \to B$, the following are equivalent.*

(i). *$F$ is an isomorphism of precategories.*
(ii). *There exist $G : B \to A$ and $\eta : 1_A = GF$ and $\epsilon : FG = 1_B$ such that*

$$\mathsf{ap}_{(\lambda H.\, FH)}(\eta) = \mathsf{ap}_{(\lambda K.\, KF)}(\epsilon^{-1}). \tag{9.4.10}$$

(iii). *There merely exist $G : B \to A$ and $\eta : 1_A = GF$ and $\epsilon : FG = 1_B$.*

*Proof.* First note that since hom-sets are sets, equalities between equalities of functors are uniquely determined by their object-parts. Thus, by function extensionality, (9.4.10) is equivalent to

$$(F_0)(\eta_0)_a = (\epsilon_0)^{-1}{}_{F_0 a}. \tag{9.4.11}$$

for all $a : A_0$. Note that this is precisely the triangle identity for $G_0$, $\eta_0$, and $\epsilon_0$ to be a proof that $F_0$ is a half adjoint equivalence of types.
Now suppose (i). Let $G_0 : B_0 \to A_0$ be the inverse of $F_0$, with $\eta_0 : \mathsf{id}_{A_0} = G_0 F_0$ and $\epsilon_0 : F_0 G_0 = \mathsf{id}_{B_0}$ satisfying the triangle identity, which is precisely (9.4.11). Now define $G_{b,b'} : \hom_B(b, b') \to \hom_A(G_0 b, G_0 b')$ by

$$G_{b,b'}(g) :\equiv (F_{G_0 b, G_0 b'})^{-1}\Big(\mathsf{idtoiso}((\epsilon_0)^{-1}{}_{b'}) \circ g \circ \mathsf{idtoiso}((\epsilon_0)_b)\Big)$$

(using the assumption that $F$ is fully faithful). Since $\mathsf{idtoiso}$ takes inverses to inverses and concatenation to composition, and $F$ is a functor, it follows that $G$ is a functor.
By definition, we have $(GF)_0 \equiv G_0 F_0$, which is equal to $\mathsf{id}_{A_0}$ by $\eta_0$. To obtain $1_A = GF$, we need to show that when transported along $\eta_0$, the identity function of $\hom_A(a, a')$ becomes equal to the composite $G_{Fa,Fa'} \circ F_{a,a'}$. In other words, for any $f : \hom_A(a, a')$ we must have

$$\begin{aligned}
&\mathsf{idtoiso}((\eta_0)_{a'}) \circ f \circ \mathsf{idtoiso}((\eta_0)^{-1}{}_a) \\
&\quad = (F_{GFa,GFa'})^{-1}\Big(\mathsf{idtoiso}((\epsilon_0)^{-1}{}_{Fa'}) \circ F_{a,a'}(f) \circ \mathsf{idtoiso}((\epsilon_0)_{Fa})\Big).
\end{aligned}$$

But this is equivalent to

$$\begin{aligned}
&(F_{GFa,GFa'})\Big(\mathsf{idtoiso}((\eta_0)_{a'}) \circ f \circ \mathsf{idtoiso}((\eta_0)^{-1}{}_a)\Big) \\
&\qquad = \mathsf{idtoiso}((\epsilon_0)^{-1}{}_{Fa'}) \circ F_{a,a'}(f) \circ \mathsf{idtoiso}((\epsilon_0)_{Fa}).
\end{aligned}$$

which follows from functoriality of $F$, the fact that $F$ preserves $\mathsf{idtoiso}$, and (9.4.11). Thus we have $\eta : 1_A = GF$.
On the other side, we have $(FG)_0 \equiv F_0 G_0$, which is equal to $\mathsf{id}_{B_0}$ by $\epsilon_0$. To obtain $FG = 1_B$, we need to show that when transported along $\epsilon_0$, the identity function of $\hom_B(b, b')$ becomes equal to the composite $F_{Gb,Gb'} \circ G_{b,b'}$. That is, for any $g : \hom_B(b, b')$ we must have

$$\begin{aligned}
&F_{Gb,Gb'}\Big((F_{Gb,Gb'})^{-1}\Big(\mathsf{idtoiso}((\epsilon_0)^{-1}{}_{b'}) \circ g \circ \mathsf{idtoiso}((\epsilon_0)_b)\Big)\Big) \\
&\qquad = \mathsf{idtoiso}((\epsilon_0{}^{-1})_{b'}) \circ g \circ \mathsf{idtoiso}((\epsilon_0)_b).
\end{aligned}$$

But this is just the fact that $(F_{Gb,Gb'})^{-1}$ is the inverse of $F_{Gb,Gb'}$. And we have remarked that (9.4.10) is equivalent to (9.4.11), so (ii) holds.
Conversely, suppose given (ii); then the object-parts of $G$, $\eta$, and $\epsilon$ together with (9.4.11) show that $F_0$ is an equivalence of types. And for $a, a' : A_0$, we define $\overline{G}_{a,a'} : \hom_B(Fa, Fa') \to \hom_A(a, a')$ by

$$\overline{G}_{a,a'}(g) :\equiv \mathsf{idtoiso}(\eta^{-1})_{a'} \circ G(g) \circ \mathsf{idtoiso}(\eta)_a. \tag{9.4.12}$$

By naturality of $\mathsf{idtoiso}(\eta)$, for any $f : \hom_A(a, a')$ we have

$$\begin{aligned}
\overline{G}_{a,a'}(F_{a,a'}(f)) &= \mathsf{idtoiso}(\eta^{-1})_{a'} \circ G(F(f)) \circ \mathsf{idtoiso}(\eta)_a \\
&= \mathsf{idtoiso}(\eta^{-1})_{a'} \circ \mathsf{idtoiso}(\eta)_{a'} \circ f \\
&= f.
\end{aligned}$$

On the other hand, for $g : \hom_B(Fa, Fa')$ we have

$$\begin{aligned}
F_{a,a'}(\overline{G}_{a,a'}(g)) &= F(\mathsf{idtoiso}(\eta^{-1})_{a'}) \circ F(G(g)) \circ F(\mathsf{idtoiso}(\eta)_a) \\
&= \mathsf{idtoiso}(\epsilon)_{Fa'} \circ F(G(g)) \circ \mathsf{idtoiso}(\epsilon^{-1})_{Fa} \\
&= \mathsf{idtoiso}(\epsilon)_{Fa'} \circ \mathsf{idtoiso}(\epsilon^{-1})_{Fa'} \circ g \\
&= g.
\end{aligned}$$

(There are lemmas needed here regarding the compatibility of idtoiso and whiskering, which we leave it to the reader to state and prove.)
Thus, $F_{a,a'}$ is an equivalence, so $F$ is fully faithful; i.e. (i) holds.
Now the composite (i)→(ii)→(i) is equal to the identity since (i) is a mere proposition. On the other side, tracing through the above constructions we see that the composite (ii)→(i)→(ii) essentially preserves the object-parts $G_0$, $\eta_0$, $\epsilon_0$, and the object-part of (9.4.10). And in the latter three cases, the object-part is all there is, since hom-sets are sets.
Thus, it suffices to show that we recover the action of $G$ on hom-sets. In other words, we must show that if $g : \hom_B(b, b')$, then

$$G_{b,b'}(g) = \overline{G}_{G_0 b, G_0 b'}\Big(\mathsf{idtoiso}((\epsilon_0)^{-1}{}_{b'}) \circ g \circ \mathsf{idtoiso}((\epsilon_0)_b)\Big)$$

where $\overline{G}$ is defined by (9.4.12). However, this follows from functoriality of $G$ and the *other* triangle identity, which we have seen in **??** is equivalent to (9.4.11).
Now since (i) is a mere proposition, so is (ii), so it suffices to show they are logically equivalent to (iii). Of course, (ii)→(iii), so let us assume (iii). Since (i) is a mere proposition, we may assume given $G$, $\eta$, and $\epsilon$. Then $G_0$ along with $\eta$ and $\epsilon$ imply that $F_0$ is an equivalence. Moreover, we also have natural isomorphisms $\mathsf{idtoiso}(\eta) : 1_A \cong GF$ and $\mathsf{idtoiso}(\epsilon) : FG \cong 1_B$, so by Lemma 9.4.2, $F$ is an equivalence of precategories, and in particular fully faithful. □

*Example* 9.4.13. Let $X$ be a type and $x_0 : X$ an element, and let $X_{\mathrm{ch}}$ denote the *chaotic* or *indiscrete* precategory on $X$. By definition, we have $(X_{\mathrm{ch}})_0 :\equiv X$, and $\hom_{X_{\mathrm{ch}}}(x, x') :\equiv \mathbf{1}$ for all $x, x'$. Then the unique functor $X_{\mathrm{ch}} \to \mathbf{1}$ is an equivalence of precategories, but not an isomorphism unless $X$ is contractible.
This example also shows that a precategory can be equivalent to a category without itself being a category. Of course, if a precategory is *isomorphic* to a category, then it must itself be a category.

**Lemma 9.4.14.** *For categories $A$ and $B$, a functor $F : A \to B$ is an equivalence of categories if and only if it is an isomorphism of categories.*

**Lemma 9.4.15.** *If $A$ and $B$ are precategories, then the function*

$$(A = B) \to (A \cong B)$$

*(defined by induction from the identity functor) is an equivalence of types.*

**Theorem 9.4.16.** *If $A$ and $B$ are categories, then the function*

$$(A = B) \to (A \simeq B)$$

*(defined by induction from the identity functor) is an equivalence of types.*

## 9.5 The Yoneda lemma

**Definition 9.5.1.** For a precategory $A$, its **opposite** $A^{\mathrm{op}}$ is a precategory with the same type of objects, with $\hom_{A^{\mathrm{op}}}(a, b) :\equiv \hom_A(b, a)$, and with identities and composition inherited from $A$.

**Definition 9.5.2.** For precategories $A$ and $B$, their **product** $A \times B$ is a precategory with $(A \times B)_0 :\equiv A_0 \times B_0$ and

$$\hom_{A\times B}((a, b), (a', b')) :\equiv \hom_A(a, a') \times \hom_B(b, b').$$

Identities are defined by $1_{(a,b)} :\equiv (1_a, 1_b)$ and composition by $(g, g')(f, f') :\equiv ((gf), (g'f'))$.

**Lemma 9.5.3.** *For precategories $A, B, C$, the following types are equivalent.*

*(i). Functors $A \times B \to C$.*
*(ii). Functors $A \to C^B$.*

Now for any precategory $A$, we have a hom-functor

$$\hom_A : A^{\mathrm{op}} \times A \to \mathcal{S}et.$$

It takes a pair $(a, b) : (A^{\mathrm{op}})_0 \times A_0 \equiv A_0 \times A_0$ to the set $\hom_A(a, b)$. For a morphism $(f, f') : \hom_{A^{\mathrm{op}}\times A}((a, b), (a', b'))$, by definition we have $f : \hom_A(a', a)$ and $f' : \hom_A(b, b')$, so we can define

$$\begin{aligned}(\hom_A)_{(a,b),(a',b')}(f, f') &:\equiv (g \mapsto (f'gf)) \\ &: \hom_A(a, b) \to \hom_A(a', b').\end{aligned}$$

Functoriality is easy to check.

**Theorem 9.5.4** (The Yoneda lemma)**.** *For any precategory $A$, any $a : A$, and any functor $F : \mathcal{S}et^{A^{\mathrm{op}}}$, we have an isomorphism*

$$\hom_{\mathcal{S}et^{A^{\mathrm{op}}}}(\mathbf{y}a, F) \cong Fa. \qquad (9.5.5)$$

*Moreover, this is natural in both $a$ and $F$.*

**Corollary 9.5.6.** *The Yoneda embedding $\mathbf{y} : A \to \mathcal{S}et^{A^{\mathrm{op}}}$ is fully faithful.*

**Corollary 9.5.7.** *If $A$ is a category, then $\mathbf{y}_0 : A_0 \to (\mathcal{S}et^{A^{\mathrm{op}}})_0$ is an embedding. In particular, if $\mathbf{y}a = \mathbf{y}b$, then $a = b$.*

**Definition 9.5.8.** A functor $F : \mathcal{S}et^{A^{\mathrm{op}}}$ is said to be **representable** if there exists $a : A$ and an isomorphism $\mathbf{y}a \cong F$.

**Theorem 9.5.9.** *If $A$ is a category, then the type "$F$ is representable" is a mere proposition.*

**Lemma 9.5.10.** *For any precategories $A$ and $B$ and a functor $F : A \to B$, the following types are equivalent.*

*(i). $F$ is a left adjoint.*
*(ii). For each $b : B$, the functor $(a \mapsto \hom_B(Fa, b))$ from $A^{\mathrm{op}}$ to $\mathcal{S}et$ is representable.*

**Corollary 9.5.11.** *[Lemma 9.3.2] If $A$ is a category and $F : A \to B$, then the type "$F$ is a left adjoint" is a mere proposition.*

## 9.6 Strict categories

**Definition 9.6.1.** A **strict category** is a precategory whose type of objects is a set.

*Example* 9.6.2. Let $A$ be a precategory and $x : A$ an object. Then there is a precategory $\mathsf{mono}(A, x)$ as follows:

- Its objects consist of an object $y : A$ and a monomorphism $m : \hom_A(y, x)$. (As usual, $m : \hom_A(y, x)$ is a **monomorphism** (or is **monic**) if $(m \circ f = m \circ g) \Rightarrow (f = g)$.)
- Its morphisms from $(y, m)$ to $(z, n)$ are arbitrary morphisms from $y$ to $z$ in $A$ (not necessarily respecting $m$ and $n$).

An equality $(y, m) = (z, n)$ of objects in $\mathsf{mono}(A, x)$ consists of an equality $p : y = z$ and an equality $p_*(m) = n$, which by Lemma 9.1.9 is equivalently an equality $m = n \circ \mathsf{idtoiso}(p)$. Since hom-sets are sets, the type of such equalities is a mere proposition. But since $m$ and $n$ are monomorphisms, the type of morphisms $f$ such that $m = n \circ f$ is also a mere proposition. Thus, if $A$ is a category, then $(y, m) = (z, n)$ is a mere proposition, and hence $\mathsf{mono}(A, x)$ is a strict category.

*Example* 9.6.3. Let $E/F$ be a finite Galois extension of fields, and $G$ its Galois group. Then there is a strict category whose objects are intermediate fields $F \subseteq K \subseteq E$, and whose morphisms are field homomorphisms which fix $F$ pointwise (but need not commute with the inclusions into $E$). There is another strict category whose objects are subgroups $H \subseteq G$, and whose morphisms are morphisms of $G$-sets $G/H \to G/K$. The fundamental theorem of Galois theory says that these two precategories are isomorphic (not merely equivalent).

## 9.7 †-categories

**Definition 9.7.1.** A **†-precategory** is a precategory $A$ together with the following.

(i). For each $x, y : A$, a function $(-)^\dagger : \hom_A(x, y) \to \hom_A(y, x)$.
(ii). For all $x : A$, we have $(1_x)^\dagger = 1_x$.
(iii). For all $f, g$ we have $(g \circ f)^\dagger = f^\dagger \circ g^\dagger$.
(iv). For all $f$ we have $(f^\dagger)^\dagger = f$.

**Definition 9.7.2.** A morphism $f : \hom_A(x, y)$ in a †-precategory is **unitary** if $f^\dagger \circ f = 1_x$ and $f \circ f^\dagger = 1_y$.

**Lemma 9.7.3.** *If $p : (x = y)$, then $\mathsf{idtoiso}(p)$ is unitary.*

**Definition 9.7.4.** A **†-category** is a †-precategory such that for all $x, y : A$, the function

$$(x = y) \to (x \cong^\dagger y)$$

from Lemma 9.7.3 is an equivalence.

*Example* 9.7.5. The category $\mathcal{R}el$ from Example 9.1.16 becomes a †-precategory if we define $(R^\dagger)(y, x) :\equiv R(x, y)$. The proof that $\mathcal{R}el$ is a category actually shows that every isomorphism is unitary; hence $\mathcal{R}el$ is also a †-category.

*Example* 9.7.6. Any groupoid becomes a †-category if we define $f^\dagger :\equiv f^{-1}$.

*Example* 9.7.7. Let $\mathcal{H}ilb$ be the following precategory.

- Its objects are finite-dimensional vector spaces equipped with an inner product $\langle -, - \rangle$.
- Its morphisms are arbitrary linear maps.

By standard linear algebra, any linear map $f : V \to W$ between finite dimensional inner product spaces has a uniquely defined adjoint $f^\dagger : W \to V$, characterized by $\langle fv, w\rangle = \langle v, f^\dagger w\rangle$. In this way, $\mathcal{H}ilb$ becomes a †-precategory. Moreover, a linear isomorphism is unitary precisely when it is an **isometry**, i.e. $\langle fv, fw\rangle = \langle v, w\rangle$. It follows from this that $\mathcal{H}ilb$ is a †-category, though it is not a category (not every linear isomorphism is unitary).

## 9.8 The structure identity principle

**Definition 9.8.1.** A **notion of structure** $(P, H)$ over $X$ consists of the following.

(i). A type family $P : X_0 \to \mathcal{U}$. For each $x : X_0$ the elements of $Px$ are called $(P, H)$**-structures** on $x$.

(ii). For $x, y : X_0$ and $\alpha : Px,\ \beta : Py$, to each $f : \hom_X(x, y)$ a mere proposition

$$H_{\alpha\beta}(f).$$

If $H_{\alpha\beta}(f)$ is true, we say that $f$ is a $(P, H)$**-homomorphism** from $\alpha$ to $\beta$.

(iii). For $x : X_0$ and $\alpha : Px$, we have $H_{\alpha\alpha}(1_x)$.

(iv). For $x, y, z : X_0$ and $\alpha : Px,\ \beta : Py,\ \gamma : Pz$, if $f : \hom_X(x, y)$ and $g : \hom_X(y, z)$, we have

$$H_{\alpha\beta}(f) \to H_{\beta\gamma}(g) \to H_{\alpha\gamma}(g \circ f).$$

When $(P, H)$ is a notion of structure, for $\alpha, \beta : Px$ we define

$$(\alpha \leq_x \beta) :\equiv H_{\alpha\beta}(1_x).$$

By (iii) and (iv), this is a preorder (Example 9.1.11) with $Px$ its type of objects. We say that $(P, H)$ is a **standard notion of structure** if this preorder is in fact a partial order, for all $x : X$.

**Theorem 9.8.2** (Structure identity principle)**.** *If $X$ is a category and $(P, H)$ is a standard notion of structure over $X$, then the precategory $\mathsf{Str}_{(P,H)}(X)$ is a category.*

*Example* 9.8.3. Let $A$ be a precategory and $B$ a category. There is a precategory $B^{A_0}$ whose objects are functions $A_0 \to B_0$, and whose set of morphisms from $F_0 : A_0 \to B_0$ to $G_0 : A_0 \to B_0$ is $\prod_{(a:A_0)} \hom_B(F_0 a, G_0 a)$. Composition and identities are inherited directly from those in $B$. It is easy to show that $\gamma : \hom_{B^{A_0}}(F_0, G_0)$ is an isomorphism exactly when each component $\gamma_a$ is an isomorphism, so that we have $(F_0 \cong G_0) \simeq \prod_{(a:A_0)}(F_0 a \cong G_0 a)$. Moreover, the map $\mathsf{idtoiso} : (F_0 = G_0) \to (F_0 \cong G_0)$ of $B^{A_0}$ is equal to the composite

$$(F_0 = G_0) \longrightarrow \prod_{a:A_0} (F_0 a = G_0 a) \longrightarrow \prod_{a:A_0} (F_0 a \cong G_0 a) \longrightarrow (F_0 \cong G_0)$$

in which the first map is an equivalence by function extensionality, the second because it is a dependent product of equivalences (since $B$ is a category), and the third as remarked above. Thus, $B^{A_0}$ is a category. Now we define a notion of structure on $B^{A_0}$ for which $P(F_0)$ is the type of operations $F : \prod_{(a,a':A_0)} \hom_A(a, a') \to \hom_B(F_0 a, F_0 a')$ which extend $F_0$ to a functor (i.e. preserve composition and identities). This is a set since each $\hom_B(-,-)$ is so. Given such $F$ and $G$, we define $\gamma : \hom_{B^{A_0}}(F_0, G_0)$ to be a homomorphism if it forms a natural transformation. In Definition 9.2.3 we essentially verified that this is a notion of structure. Moreover, if $F$ and $F'$ are both structures on $F_0$ and the identity is a natural transformation from $F$ to $F'$, then for any $f : \hom_A(a, a')$ we have $F'f = F'f \circ 1_{F_0 a} = 1_{F_0 a} \circ Ff = Ff$. Applying function extensionality, we conclude $F = F'$. Thus, we have a *standard* notion of structure, and so by Theorem 9.8.2, the precategory $B^A$ is a category.

**Definition 9.8.4.**

(i). For each $\mathcal{U}$-small set $x$ define

$$Px :\equiv P_0 x \times P_1 x.$$

Here

$$P_0 x :\equiv \prod_{\omega:\Omega_0} x^{|\omega|} \to x, \text{ and}$$

$$P_1 x :\equiv \prod_{\omega:\Omega_1} x^{|\omega|} \to \mathsf{Prop}_{\mathcal{U}},$$

(ii). For $\mathcal{U}$-small sets $x, y$ and $\alpha : P^\omega x,\ \beta : P^\omega y,\ f : x \to y$, define

$$H_{\alpha\beta}(f) :\equiv H_{0,\alpha\beta}(f) \wedge H_{1,\alpha\beta}(f).$$

Here

$$H_{0,\alpha\beta}(f) :\equiv \forall(\omega : \Omega_0).\forall(u : x^{|\omega|}).\, f(\alpha u) = \beta(f \circ u), \text{ and}$$
$$H_{1,\alpha\beta}(f) :\equiv \forall(\omega : \Omega_1).\forall(u : x^{|\omega|}).\, \alpha u \to \beta(f \circ u).$$

## 9.9 The Rezk completion

**Lemma 9.9.1.** *If $A, B, C$ are precategories and $H : A \to B$ is an essentially surjective functor, then $(- \circ H) : C^B \to C^A$ is faithful.*

**Lemma 9.9.2.** *If $A, B, C$ are precategories and $H : A \to B$ is essentially surjective and full, then $(- \circ H) : C^B \to C^A$ is fully faithful.*

**Theorem 9.9.3.** *If $A, B$ are precategories, $C$ is a category, and $H : A \to B$ is a weak equivalence, then $(- \circ H) : C^B \to C^A$ is an isomorphism.*

Therefore, if a precategory $A$ admits a weak equivalence functor $A \to \widehat{A}$ into a category, then that is its "reflection" into categories: any functor from $A$ into a category will factor essentially uniquely through $\widehat{A}$. We now give two constructions of such a weak equivalence.

**Theorem 9.9.4.** *For any precategory $A$, there is a category $\widehat{A}$ and a weak equivalence $A \to \widehat{A}$.*

*Example* 9.9.5. Recall from Example 9.1.14 that for any type $X$ there is a pregroupoid with $X$ as its type of objects and $\hom(x, y) :\equiv \|x = y\|_0$. Its Rezk completion is the *fundamental groupoid* of $X$. Recalling that groupoids are equivalent to 1-types, it is not hard to identify this groupoid with $\|X\|_1$.

*Example* 9.9.6. Recall from Example 9.1.15 that there is a precategory whose type of objects is $\mathcal{U}$ and with $\hom(X, Y) :\equiv \|X \to Y\|_0$. Its Rezk completion may be called the **homotopy category of types**. Its type of objects can be identified with $\|\mathcal{U}\|_1$ (see **??**).

**Theorem 9.9.7.** *A precategory $C$ is a category if and only if for every weak equivalence of precategories $H : A \to B$, the induced functor $(- \circ H) : C^B \to C^A$ is an isomorphism of precategories.*

# Homotopy Type Theory

Formal type theory

## A.1 The first presentation

**Convertibility** $t \downarrow t'$ between terms $t$ and $t'$ is the equivalence relation generated by the defining equations for constants, the computation rule

### A.1.1 Type universes

We postulate a hierarchy of **universes** denoted by primitive constants

$$\mathcal{U}_0, \quad \mathcal{U}_1, \quad \mathcal{U}_2, \quad \ldots$$

The first two rules for universes say that they form a cumulative hierarchy of types:

- $\mathcal{U}_m : \mathcal{U}_n$ for $m < n$,
- if $A : \mathcal{U}_m$ and $m \leq n$, then $A : \mathcal{U}_n$,

and the third expresses the idea that an object of a universe can serve as a type and stand to the right of a colon in judgments:

- if $\Gamma \vdash A : \mathcal{U}_n$, and $x$ is a new variable,[1] then $\vdash (\Gamma, x : A)$ ctx.

In the body of the book, an equality judgment $A \equiv B : \mathcal{U}_n$ between types $A$ and $B$ is usually abbreviated to $A \equiv B$. This is an instance of typical ambiguity, as we can always switch to a larger universe, which however does not affect the validity of the judgment.
The following conversion rule allows us to replace a type by one equal to it in a typing judgment:

- if $a : A$ and $A \equiv B$ then $a : B$.

### A.1.2 Dependent function types ($\Pi$-types)

We introduce a primitive constant $c_\Pi$, but write $c_\Pi(A, \lambda x. B)$ as $\prod_{(x:A)} B$. Judgments concerning such expressions and expressions of the form $\lambda x. b$ are introduced by the following rules:

- if $\Gamma \vdash A : \mathcal{U}_n$ and $\Gamma, x : A \vdash B : \mathcal{U}_n$, then $\Gamma \vdash \prod_{(x:A)} B : \mathcal{U}_n$
- if $\Gamma, x : A \vdash b : B$ then $\Gamma \vdash (\lambda x. b) : (\prod_{(x:A)} B)$
- if $\Gamma \vdash g : \prod_{(x:A)} B$ and $\Gamma \vdash t : A$ then $\Gamma \vdash g(t) : B[t/x]$

If $x$ does not occur freely in $B$, we abbreviate $\prod_{(x:A)} B$ as the non-dependent function type $A \to B$ and derive the following rule:

- if $\Gamma \vdash g : A \to B$ and $\Gamma \vdash t : A$ then $\Gamma \vdash g(t) : B$

Using non-dependent function types and leaving implicit the context $\Gamma$, the rules above can be written in the following alternative style that we use in the rest of this section of the appendix:

- if $A : \mathcal{U}_n$ and $B : A \to \mathcal{U}_n$, then $\prod_{(x:A)} B(x) : \mathcal{U}_n$
- if $x : A \vdash b : B$ then $\lambda x. b : \prod_{(x:A)} B(x)$
- if $g : \prod_{(x:A)} B(x)$ and $t : A$ then $g(t) : B(t)$

[1] By "new" we mean that it does not appear in $\Gamma$ or $A$.

### A.1.3 Dependent pair types ($\Sigma$-types)

We introduce primitive constants $c_\Sigma$ and $c_{\mathsf{pair}}$. An expression of the form $c_\Sigma(A, \lambda a. B)$ is written as $\sum_{(a:A)} B$, and an expression of the form $c_{\mathsf{pair}}(a, b)$ is written as $(a, b)$. We write $A \times B$ instead of $\sum_{(x:A)} B$ if $x$ is not free in $B$.
Judgments concerning such expressions are introduced by the following rules:

- if $A : \mathcal{U}_n$ and $B : A \to \mathcal{U}_n$, then $\sum_{(x:A)} B(x) : \mathcal{U}_n$
- if, in addition, $a : A$ and $b : B(a)$, then $(a, b) : \sum_{(x:A)} B(x)$

If we have $A$ and $B$ as above, $C : (\sum_{(x:A)} B(x)) \to \mathcal{U}_m$, and

$$d : \textstyle\prod_{(x:A)} \prod_{(y:B(x))} C((x, y))$$

we can introduce a defined constant

$$f : \textstyle\prod_{(p:\sum_{(x:A)} B(x))} C(p)$$

with the defining equation

$$f((x, y)) :\equiv d(x, y).$$

Note that $C$, $d$, $x$, and $y$ may contain extra implicit parameters $x_1, \ldots, x_n$ if they were obtained in some non-empty context; therefore, the fully explicit recursion schema is

$$f(x_1, \ldots, x_n, (x(x_1, \ldots, x_n), y(x_1, \ldots, x_n))) :\equiv d(x_1, \ldots, x_n, (x(x_1, \ldots, x_n), y(x_1, \ldots, x_n))).$$

### A.1.4 Coproduct types

We introduce primitive constants $c_+$, $c_{\mathsf{inl}}$, and $c_{\mathsf{inr}}$. We write $A + B$ instead of $c_+(A, B)$, $\mathsf{inl}(a)$ instead of $c_{\mathsf{inl}}(a)$, and $\mathsf{inr}(a)$ instead of $c_{\mathsf{inr}}(a)$:

- if $A, B : \mathcal{U}_n$ then $A + B : \mathcal{U}_n$
- moreover, $\mathsf{inl} : A \to A + B$ and $\mathsf{inr} : B \to A + B$

If we have $A$ and $B$ as above, $C : A + B \to \mathcal{U}_m$, $d : \prod_{(x:A)} C(\mathsf{inl}(x))$, and $e : \prod_{(y:B)} C(\mathsf{inr}(y))$, then we can introduce a defined constant $f : \prod_{(z:A+B)} C(z)$ with the defining equations

$$f(\mathsf{inl}(x)) :\equiv d(x) \qquad \text{and} \qquad f(\mathsf{inr}(y)) :\equiv e(y).$$

### A.1.5 The finite types

We introduce primitive constants $\star$, $\mathbf{0}$, $\mathbf{1}$, satisfying the following rules:

- $\mathbf{0} : \mathcal{U}_0$, $\mathbf{1} : \mathcal{U}_0$
- $\star : \mathbf{1}$

Given $C : \mathbf{0} \to \mathcal{U}_n$ we can introduce a defined constant $f : \prod_{(x:\mathbf{0})} C(x)$, with no defining equations.
Given $C : \mathbf{1} \to \mathcal{U}_n$ and $d : C(\star)$ we can introduce a defined constant $f : \prod_{(x:\mathbf{1})} C(x)$, with defining equation $f(\star) :\equiv d$.

### A.1.6 Natural numbers

The type of natural numbers is obtained by introducing primitive constants $\mathbb{N}$, $0$, and $\mathsf{succ}$ with the following rules:

- $\mathbb{N} : \mathcal{U}_0$,
- $0 : \mathbb{N}$,
- $\mathsf{succ} : \mathbb{N} \to \mathbb{N}$.

Furthermore, we can define functions by primitive recursion. If we have $C : \mathbb{N} \to \mathcal{U}_k$ we can introduce a defined constant $f : \prod_{(x:\mathbb{N})} C(x)$ whenever we have

$$d : C(0)$$
$$e : \textstyle\prod_{(x:\mathbb{N})} (C(x) \to C(\mathsf{succ}(x)))$$

with the defining equations

$$f(0) :\equiv d \qquad \text{and} \qquad f(\mathsf{succ}(x)) :\equiv e(x, f(x)).$$

### A.1.7 $W$-types

For $W$-types we introduce primitive constants $c_{\mathsf{W}}$ and $c_{\mathsf{sup}}$. An expression of the form $c_{\mathsf{W}}(A, \lambda x. B)$ is written as $\mathsf{W}_{(x:A)} B$, and an expression of the form $c_{\mathsf{sup}}(x, u)$ is written as $\mathsf{sup}(x, u)$:

- if $A : \mathcal{U}_n$ and $B : A \to \mathcal{U}_n$, then $\mathsf{W}_{(x:A)} B(x) : \mathcal{U}_n$
- if moreover, $a : A$ and $u : B(a) \to \mathsf{W}_{(x:A)} B(x)$ then $\mathsf{sup}(a, u) : \mathsf{W}_{(x:A)} B(x)$.

Here also we can define functions by total recursion. If we have $A$ and $B$ as above and $C : (\mathsf{W}_{(x:A)} B(x)) \to \mathcal{U}_m$, then we can introduce a defined constant $f : \prod_{(z:\mathsf{W}_{(x:A)} B(x))} C(z)$ whenever we have

$$d : \textstyle\prod_{(a:A)} \prod_{(u:B(a) \to \mathsf{W}_{(x:A)} B(x))} ((\prod_{(y:B(a))} C(u(y))) \to C(\mathsf{sup}(a, u)))$$

with the defining equation

$$f(\mathsf{sup}(a, u)) :\equiv d(a, u, f \circ u).$$

### A.1.8 Identity types

We introduce primitive constants $c_=$ and $c_{\mathsf{refl}}$. We write $a =_A b$ for $c_=(A, a, b)$ and $\mathsf{refl}_a$ for $c_{\mathsf{refl}}(A, a)$, when $a : A$ is understood:

- If $A : \mathcal{U}_n$, $a : A$, and $b : A$ then $a =_A b : \mathcal{U}_n$.
- If $a : A$ then $\mathsf{refl}_a : a =_A a$.

Given $a : A$, if $y : A, z : a =_A y \vdash C : \mathcal{U}_m$ and $\vdash d : C[a, \mathsf{refl}_a / y, z]$ then we can introduce a defined constant

$$f : \textstyle\prod_{(y:A)} \prod_{(z:a =_A y)} C$$

with defining equation

$$f(a, \mathsf{refl}_a) :\equiv d.$$

## A.2 The second presentation

In this section, there are three kinds of judgments

$$\Gamma\ \mathsf{ctx} \qquad\qquad \Gamma \vdash a : A \qquad\qquad \Gamma \vdash a \equiv a' : A$$

which we specify by providing inference rules for deriving them. A typical **inference rule** has the form

$$\frac{\mathcal{J}_1 \qquad \cdots \qquad \mathcal{J}_k}{\mathcal{J}}\ \textsc{Name}$$

It says that we may derive the **conclusion** $\mathcal{J}$, provided that we have already derived the **hypotheses** $\mathcal{J}_1, \ldots, \mathcal{J}_k$.
A **derivation** of a judgment is a tree constructed from such inference rules, with the judgment at the root of the tree.

### A.2.1 Contexts

A context is a list

$$x_1{:}A_1, x_2{:}A_2, \ldots, x_n{:}A_n$$

The judgment $\Gamma$ ctx formally expresses the fact that $\Gamma$ is a well-formed context, and is governed by the rules of inference

$$\frac{}{\cdot\ \mathsf{ctx}}\ \mathsf{ctx}\text{-}\textsc{emp} \qquad\qquad \frac{x_1{:}A_1, \ldots, x_{n-1}{:}A_{n-1} \vdash A_n : \mathcal{U}_i}{(x_1{:}A_1, \ldots, x_n{:}A_n)\ \mathsf{ctx}}\ \mathsf{ctx}\text{-}\textsc{ext}$$

### A.2.2 Structural rules

$$\frac{(x_1{:}A_1, \ldots, x_n{:}A_n)\ \mathsf{ctx}}{x_1{:}A_1, \ldots, x_n{:}A_n \vdash x_i : A_i}\ \mathsf{Vble}$$

The following important principles, called **substitution** and **weakening**, need not be explicitly assumed. For the typing judgments these principles are manifested as

$$\frac{\Gamma \vdash a : A \qquad \Gamma, x{:}A, \Delta \vdash b : B}{\Gamma, \Delta[a/x] \vdash b[a/x] : B[a/x]}\ \mathsf{Subst}_1$$

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma, \Delta \vdash b : B}{\Gamma, x{:}A, \Delta \vdash b : B}\ \mathsf{Wkg}_1$$

and for judgmental equalities they become

$$\frac{\Gamma \vdash a : A \qquad \Gamma, x{:}A, \Delta \vdash b \equiv c : B}{\Gamma, \Delta[a/x] \vdash b[a/x] \equiv c[a/x] : B[a/x]}\ \mathsf{Subst}_2$$

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma, \Delta \vdash b \equiv c : B}{\Gamma, x{:}A, \Delta \vdash b \equiv c : B}\ \mathsf{Wkg}_2$$

In addition to the judgmental equality rules given for each type former, we also assume that judgmental equality is an equivalence relation respected by typing.

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash a \equiv a : A} \qquad \frac{\Gamma \vdash a \equiv b : A}{\Gamma \vdash b \equiv a : A} \qquad \frac{\Gamma \vdash a \equiv b : A \qquad \Gamma \vdash b \equiv c : A}{\Gamma \vdash a \equiv c : A}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash A \equiv B : \mathcal{U}_i}{\Gamma \vdash a : B} \qquad \frac{\Gamma \vdash a \equiv b : A \qquad \Gamma \vdash A \equiv B : \mathcal{U}_i}{\Gamma \vdash a \equiv b : B}$$

Additionally, for all the type formers below, we assume rules stating that each constructor preserves definitional equality in each of its arguments; for instance, along with the $\Pi$-INTRO rule, we assume the rule

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma, x{:}A \vdash B : \mathcal{U}_i \qquad \Gamma, x{:}A \vdash b \equiv b' : B}{\Gamma \vdash \lambda x.\, b \equiv \lambda x.\, b' : \prod_{(x:A)} B}\ \Pi\text{-}\textsc{intro-eq}$$

However, we omit these rules for brevity.

### A.2.3 Type universes

We postulate an infinite hierarchy of type universes

$$\mathcal{U}_0, \quad \mathcal{U}_1, \quad \mathcal{U}_2, \quad \ldots$$

Each universe is contained in the next, and any type in $\mathcal{U}_i$ is also in $\mathcal{U}_{i+1}$:

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathcal{U}_i : \mathcal{U}_{i+1}}\ \mathcal{U}\text{-}\textsc{intro} \qquad\qquad \frac{\Gamma \vdash A : \mathcal{U}_i}{\Gamma \vdash A : \mathcal{U}_{i+1}}\ \mathcal{U}\text{-}\textsc{cumul}$$

### A.2.4 Dependent function types ($\Pi$-types)

$$x{:}A \vdash B : \mathcal{U}_i.$$

- a **formation rule**, stating when the type former can be applied;
- some **introduction rules**, stating how to inhabit the type;
- **elimination rules**, or an induction principle, stating how to use an element of the type;
- **computation rules**, which are judgmental equalities explaining what happens when elimination rules are applied to results of introduction rules;
- optional **uniqueness principles**, which are judgmental equalities explaining how every element of the type is uniquely determined by the results of elimination rules applied to it.

For the dependent function type these rules are:

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma, x{:}A \vdash B : \mathcal{U}_i}{\Gamma \vdash \prod_{(x:A)} B : \mathcal{U}_i}\ \Pi\text{-}\textsc{form}$$

$$\frac{\Gamma, x{:}A \vdash b : B}{\Gamma \vdash \lambda(x:A).\, b : \prod_{(x:A)} B}\ \Pi\text{-}\textsc{intro}$$

$$\frac{\Gamma \vdash f : \prod_{(x:A)} B \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B[a/x]}\ \Pi\text{-}\textsc{elim}$$

$$\frac{\Gamma, x{:}A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda(x:A).\, b)(a) \equiv b[a/x] : B[a/x]}\ \Pi\text{-}\textsc{comp}$$

$$\frac{\Gamma \vdash f : \prod_{(x:A)} B}{\Gamma \vdash f \equiv (\lambda x.\, f(x)) : \prod_{(x:A)} B}\ \Pi\text{-}\textsc{uniq}$$

### A.2.5 Dependent pair types ($\Sigma$-types)

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma, x{:}A \vdash B : \mathcal{U}_i}{\Gamma \vdash \sum_{(x:A)} B : \mathcal{U}_i}\ \Sigma\text{-}\textsc{form}$$

$$\frac{\Gamma, x{:}A \vdash B : \mathcal{U}_i \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B[a/x]}{\Gamma \vdash (a, b) : \sum_{(x:A)} B}\ \Sigma\text{-}\textsc{intro}$$

$$\frac{\begin{array}{c}\Gamma, z{:}\sum_{(x:A)} B \vdash C : \mathcal{U}_i \\ \Gamma, x{:}A, y{:}B \vdash g : C[(x,y)/z] \qquad \Gamma \vdash p : \sum_{(x:A)} B\end{array}}{\Gamma \vdash \mathsf{ind}_{\sum_{(x:A)} B}(z.C, x.y.g, p) : C[p/z]}\ \Sigma\text{-}\textsc{elim}$$

$$\frac{\begin{array}{c}\Gamma, z{:}\sum_{(x:A)} B \vdash C : \mathcal{U}_i \qquad \Gamma, x{:}A, y{:}B \vdash g : C[(x,y)/z] \\ \Gamma \vdash a : A \qquad \Gamma \vdash b : B[a/x]\end{array}}{\Gamma \vdash \mathsf{ind}_{\sum_{(x:A)} B}(z.C, x.y.g, (a,b)) \equiv g[a, b/x, y] : C[(a,b)/z]}\ \Sigma\text{-}\textsc{comp}$$

### A.2.6 Coproduct types

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash B : \mathcal{U}_i}{\Gamma \vdash A + B : \mathcal{U}_i}\ +\text{-}\textsc{form}$$

$$\frac{\begin{array}{c}\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash B : \mathcal{U}_i \\ \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathsf{inl}(a) : A + B}\ +\text{-}\textsc{intro}_1$$

$$\frac{\begin{array}{c}\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash B : \mathcal{U}_i \\ \Gamma \vdash b : B\end{array}}{\Gamma \vdash \mathsf{inr}(b) : A + B}\ +\text{-}\textsc{intro}_2$$

$$\frac{\begin{array}{c}\Gamma, z{:}(A+B) \vdash C : \mathcal{U}_i \\ \Gamma, x{:}A \vdash c : C[\mathsf{inl}(x)/z] \qquad \Gamma, y{:}B \vdash d : C[\mathsf{inr}(y)/z] \\ \Gamma \vdash e : A + B\end{array}}{\Gamma \vdash \mathsf{ind}_{A+B}(z.C, x.c, y.d, e) : C[e/z]}\ +\text{-}\textsc{elim}$$

$$\frac{\begin{array}{c}\Gamma, z{:}(A+B) \vdash C : \mathcal{U}_i \\ \Gamma, x{:}A \vdash c : C[\mathsf{inl}(x)/z] \qquad \Gamma, y{:}B \vdash d : C[\mathsf{inr}(y)/z] \\ \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathsf{ind}_{A+B}(z.C, x.c, y.d, \mathsf{inl}(a)) \equiv c[a/x] : C[\mathsf{inl}(a)/z]}\ +\text{-}\textsc{comp}_1$$

$$\frac{\begin{array}{c}\Gamma, z{:}(A+B) \vdash C : \mathcal{U}_i \\ \Gamma, x{:}A \vdash c : C[\mathsf{inl}(x)/z] \qquad \Gamma, y{:}B \vdash d : C[\mathsf{inr}(y)/z] \\ \Gamma \vdash b : B\end{array}}{\Gamma \vdash \mathsf{ind}_{A+B}(z.C, x.c, y.d, \mathsf{inr}(b)) \equiv d[b/y] : C[\mathsf{inr}(b)/z]}\ +\text{-}\textsc{comp}_2$$

### A.2.7 The empty type $\mathbf{0}$

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathbf{0} : \mathcal{U}_i}\ \mathbf{0}\text{-}\textsc{form} \qquad\qquad \frac{\Gamma, x{:}\mathbf{0} \vdash C : \mathcal{U}_i \qquad \Gamma \vdash a : \mathbf{0}}{\Gamma \vdash \mathsf{ind}_{\mathbf{0}}(x.C, a) : C[a/x]}\ \mathbf{0}\text{-}\textsc{elim}$$

### A.2.8 The unit type $\mathbf{1}$

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathbf{1} : \mathcal{U}_i}\ \mathbf{1}\text{-FORM} \qquad \frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \star : \mathbf{1}}\ \mathbf{1}\text{-INTRO}$$

$$\frac{\Gamma, x{:}\mathbf{1} \vdash C : \mathcal{U}_i \qquad \Gamma \vdash c : C[\star/x] \qquad \Gamma \vdash a : \mathbf{1}}{\Gamma \vdash \mathsf{ind}_{\mathbf{1}}(x.C, c, a) : C[a/x]}\ \mathbf{1}\text{-ELIM}$$

$$\frac{\Gamma, x{:}\mathbf{1} \vdash C : \mathcal{U}_i \qquad \Gamma \vdash c : C[\star/x]}{\Gamma \vdash \mathsf{ind}_{\mathbf{1}}(x.C, c, \star) \equiv c : C[\star/x]}\ \mathbf{1}\text{-COMP}$$

### A.2.9 The natural number type

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathbb{N} : \mathcal{U}_i}\ \mathbb{N}\text{-FORM} \qquad \frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash 0 : \mathbb{N}}\ \mathbb{N}\text{-INTRO}_1$$

$$\frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{succ}(n) : \mathbb{N}}\ \mathbb{N}\text{-INTRO}_2$$

$$\frac{\begin{array}{c}\Gamma, x{:}\mathbb{N} \vdash C : \mathcal{U}_i \qquad \Gamma \vdash c_0 : C[0/x] \\ \Gamma, x{:}\mathbb{N}, y{:}C \vdash c_s : C[\mathsf{succ}(x)/x] \qquad \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, n) : C[n/x]}\ \mathbb{N}\text{-ELIM}$$

$$\frac{\begin{array}{c}\Gamma, x{:}\mathbb{N} \vdash C : \mathcal{U}_i \\ \Gamma \vdash c_0 : C[0/x] \qquad \Gamma, x{:}\mathbb{N}, y{:}C \vdash c_s : C[\mathsf{succ}(x)/x]\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, 0) \equiv c_0 : C[0/x]}\ \mathbb{N}\text{-COMP}_1$$

$$\frac{\begin{array}{c}\Gamma, x{:}\mathbb{N} \vdash C : \mathcal{U}_i \qquad \Gamma \vdash c_0 : C[0/x] \\ \Gamma, x{:}\mathbb{N}, y{:}C \vdash c_s : C[\mathsf{succ}(x)/x] \qquad \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, \mathsf{succ}(n))}\ \mathbb{N}\text{-COMP}_2$$
$$\equiv c_s[n, \mathsf{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, n)/x, y] : C[\mathsf{succ}(n)/x]$$

### A.2.10 Identity types

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : A}{\Gamma \vdash a =_A b : \mathcal{U}_i}\ {=}\text{-FORM}$$

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{refl}_a : a =_A a}\ {=}\text{-INTRO}$$

$$\frac{\begin{array}{c}\Gamma, x{:}A, y{:}A, p{:}x =_A y \vdash C : \mathcal{U}_i \\ \Gamma, z{:}A \vdash c : C[z, z, \mathsf{refl}_z/x, y, p] \\ \Gamma \vdash a : A \qquad \Gamma \vdash b : A \qquad \Gamma \vdash p' : a =_A b\end{array}}{\Gamma \vdash \mathsf{ind}_{=_A}(x.y.p.C, z.c, a, b, p') : C[a, b, p'/x, y, p]}\ {=}\text{-ELIM}$$

$$\frac{\begin{array}{c}\Gamma, x{:}A, y{:}A, p{:}x =_A y \vdash C : \mathcal{U}_i \\ \Gamma, z{:}A \vdash c : C[z, z, \mathsf{refl}_z/x, y, p] \qquad \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathsf{ind}_{=_A}(x.y.p.C, z.c, a, a, \mathsf{refl}_a) \equiv c[a/z] : C[a, a, \mathsf{refl}_a/x, y, p]}\ {=}\text{-COMP}$$

## A.3 Homotopy type theory

### A.3.1 Function extensionality and univalence

Axiom 2.9.3 is formalized by introduction of a constant funext which asserts that happly is an equivalence:

$$\frac{\Gamma \vdash f : \prod_{(x:A)} B \qquad \Gamma \vdash g : \prod_{(x:A)} B}{\Gamma \vdash \mathsf{funext}(f, g) : \mathsf{isequiv}(\mathsf{happly}_{f,g})}\ \Pi\text{-EXT}$$

The definitions of happly and isequiv can be found in (2.9.2) and §4.5, respectively.
Axiom 2.10.3 is formalized in a similar fashion, too:

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash B : \mathcal{U}_i}{\Gamma \vdash \mathsf{univalence}(A, B) : \mathsf{isequiv}(\mathsf{idtoeqv}_{A,B})}\ \mathcal{U}_i\text{-UNIV}$$

The definition of idtoeqv can be found in (2.10.2).

### A.3.2 The circle

Here we give an example of a basic higher inductive type; others follow the same general scheme, albeit with elaborations.
Note that the rules below do not precisely follow the pattern of the ordinary inductive types in **??**: the rules refer to the notions of transport and functoriality of maps (§2.2), and the second computation rule is a propositional, not judgmental, equality. These differences are discussed in **??**.

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathbb{S}^1 : \mathcal{U}_i}\ \mathbb{S}^1\text{-FORM} \qquad \frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathsf{base} : \mathbb{S}^1}\ \mathbb{S}^1\text{-INTRO}_1$$

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathsf{loop} : \mathsf{base} =_{\mathbb{S}^1} \mathsf{base}}\ \mathbb{S}^1\text{-INTRO}_2$$

$$\frac{\begin{array}{c}\Gamma, x{:}\mathbb{S}^1 \vdash C : \mathcal{U}_i \\ \Gamma \vdash b : C[\mathsf{base}/x] \qquad \Gamma \vdash \ell : b =^{C}_{\mathsf{loop}} b \qquad \Gamma \vdash p : \mathbb{S}^1\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{S}^1}(x.C, b, \ell, p) : C[p/x]}\ \mathbb{S}^1\text{-ELIM}$$

$$\frac{\Gamma, x{:}\mathbb{S}^1 \vdash C : \mathcal{U}_i \qquad \Gamma \vdash b : C[\mathsf{base}/x] \qquad \Gamma \vdash \ell : b =^{C}_{\mathsf{loop}} b}{\Gamma \vdash \mathsf{ind}_{\mathbb{S}^1}(x.C, b, \ell, \mathsf{base}) \equiv b : C[\mathsf{base}/x]}\ \mathbb{S}^1\text{-COMP}_1$$

$$\frac{\Gamma, x{:}\mathbb{S}^1 \vdash C : \mathcal{U}_i \qquad \Gamma \vdash b : C[\mathsf{base}/x] \qquad \Gamma \vdash \ell : b =^{C}_{\mathsf{loop}} b}{\Gamma \vdash \mathbb{S}^1\text{-}\mathsf{loopcomp} : \mathsf{apd}_{(\lambda y.\, \mathsf{ind}_{\mathbb{S}^1}(x.C, b, \ell, y))}(\mathsf{loop}) = \ell}\ \mathbb{S}^1\text{-COMP}_2$$

In $\mathsf{ind}_{\mathbb{S}^1}$, $x$ is bound in $C$. The notation $b =^{C}_{\mathsf{loop}} b$ for dependent paths was introduced in Section 6.2.

## A.4 Basic metatheory

**Theorem A.4.1.** *If $A : \mathcal{U}$ and $A \downarrow A'$ then $A' : \mathcal{U}$. If $t : A$ and $t \downarrow t'$ then $t' : A$.*

**Theorem A.4.2.** *If $A : \mathcal{U}$ then $A$ is strongly normalizable. If $t : A$ then $A$ and $t$ are strongly normalizable.*

**Lemma A.4.3.** *The terms in normal form can be described by the following syntax:*

$$\begin{aligned} v &::= k \mid \lambda x.\, v \mid c(\vec{v}) \mid f(\vec{v}), \\ k &::= x \mid k(v) \mid f(\vec{v})(k), \end{aligned}$$

*where $f(\vec{v})$ represents a partial application of the defined function $f$. In particular, a type in normal form is of the form $k$ or $c(\vec{v})$.*

**Theorem A.4.4.** *If $A$ is in normal form then the judgment $A : \mathcal{U}$ is decidable. If $A : \mathcal{U}$ and $t$ is in normal form then the judgment $t : A$ is decidable.*

**Corollary A.4.5.** *The system in Appendix A.1 is logically consistent.*

**Corollary A.4.6.** *The system in Appendix A.1 has the canonicity property.*

**Corollary A.4.7.** *The property of being a proof in the system in Appendix A.1 is decidable.*