Introduction to homotopy type theory

Pierre-Louis Curien
πr^2, IRIF (INRIA, Université Paris Diderot, CNRS)

Escuela de Sciencias Informáticas, 26-30 julio 2021

References for the course

Most of the material is directly adapted from the HoTT book:

Homotopy Type Theory: Univalent Foundations of Mathematics

`https://homotopytypetheory.org/book`.

We also rely (Grothendieck equivalence, circle) on the excellent notes of Hugo Moeneclaey:

Lecture notes on synthetic homotopy theory,

available from `https://github.com/herbelin/LMFI-HoTT`.

What is type theory?

The origin is the work on solving the paradoxes in the foundations of mathematics (beginning of the twentieth century):

$$\{a \mid a \notin a\}$$

leads to contradictions! Indeed let b be this set, and suppose $b \in b$. Then by definition of b, $b \notin b$: contradiction!

Russell invented type theory to put discipline in this jungle. One can only form sets of the form $\{a \in A \mid \text{some property holds}\}$. Here A is a type.

The modern forms of type theory start with Church (λ-calculus), until its present form know as Martin-Löf dependent type theory (1970's), for which we provide an introduction in this course.

Homotopy type theory

Dependent type theory has received a new impetus in the late 2000's under the influence of Fields medallist Voevodsky, who found an interpretation of types as topological spaces.

This has led to Homotopy type theory, which we shall discuss in this course.

Some simple types known to the programmers

- `nat` is a type, and $3 : \texttt{nat}$ is an element of that type.
- $\texttt{nat} \times \texttt{nat}$ is a type, and $(3, 4)$ is an element of this type.
- $\texttt{nat} \to \texttt{nat}$ is a type and the function mapping x to $2 \times x$ is an element of this type. Standard notations are

$$\begin{array}{ll} x \mapsto 2 \times x & \text{(standard in mathematical texts)} \\ \lambda x.(2 \times x) & \text{(λ-calculus)} \end{array}$$

A function can be applied to an argument: $(\lambda x.(2 \times x))\,7 \equiv 14$
More slowly. we have

$$(\lambda x.(2 \times x))\,7 \equiv 2 \times 7$$

and the rest is primary school mathematics!

λ-calculus

The λ-calculus forms the core of functional programming languages like OCaml and Haskell:

$$M ::\equiv x \parallel \lambda x.M \parallel MM$$

Definitional equality (oriented as reduction):

$$(\lambda x.M)N \to M[x \leftarrow N]$$

Warning: renaming of bound variables may be needed! Consider $M = \lambda x.\lambda y.x + y$. Then the intended conversion/value of $M\,y\,x$ is $y + x$. But blind application of the conversion gives

$$My \to \lambda y.y + y \qquad \text{and hence } (My)x \to (\lambda y.y + y)x \to x + x$$

The capture of the red occurrence of y should be avoided. The correct reduction is (α-conversion):

$$(My)x \to (\lambda z.y + z)x \to y + x$$

The wildness of untyped λ-calculus

In λ-calculus too, strange things happen! Consider

$$\Delta = \lambda x.xx$$

(self-application is permitted!). Then we have $\Delta\Delta \to \Delta\Delta$! Slowly:

$$\Delta\Delta \equiv (\lambda x.xx)\Delta \to \Delta\Delta$$

Hence computations may not terminate, or may not even produce some interesting infinite value....

To solve this problem, Church has introduced typed λ-calculi. For example, to type xx, we need to give a type $A \to B$ to x and a type A to x The slogan is that **well-typing** guarantees **termination**!

Examples of dependent types

The type $\texttt{Fin}(n)$ (defined in the second lecture) is the type of finite sets of cardinal n. For example,

$$\texttt{Can}(n) :\equiv \{0_n, \ldots (n-1)_n\} : \texttt{Fin}(n)$$

and we can define the map $\texttt{max} := n \mapsto n_{n+1}$. We have

$$\texttt{max} : \Pi_{n:\mathbb{N}} \texttt{Can}(n+1)$$

We also have the type $\Sigma_{n:\mathbb{N}} \texttt{Can}(n+1)$, whose elements are pairs of a number n and an element of $\texttt{Can}(n+1)$. For example:

$$(4, 2_5) : \Sigma_{n:\mathbb{N}} \texttt{Can}(n+1)$$

Other example: $\texttt{List}(n, A)$ (lists of length n of elements of A), with

$$\texttt{nil} : \texttt{List}(0, A) \quad \text{and} \quad \texttt{cons} : A \times \texttt{List}(n, A) \to \texttt{List}(n+1, A)$$

Σ-types versus product types, Π-types versus function types

If B does not depend on A, then $\Sigma_{x:A}B$ is $A \times B$

(think of $A \times B$ as $B + B + \ldots + B$ for cardinal of A copies of B).

If B does not depend on A, then $\Pi_{x:A}B$ is $A \to B$, also written B^A

(think of B^A as $B \times B \times \ldots \times B$ for cardinal of A copies of B).

The star of (homotopy) type theory

The most important example of dependent type in Martin-Löf type theory is the identity type. If $x : A$ and $y : A$, then

$$\mathtt{Id}_A(x, y)\ , \quad \text{also written} \quad x =_A y\ , \text{ is a type}$$

When $x = y$, there is a term (constructor) of this type:

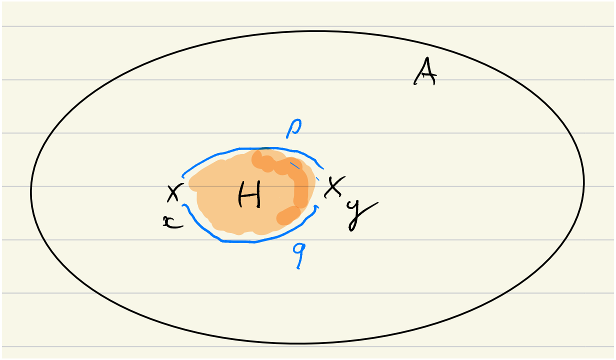
$$\mathtt{refl}(x) : x =_A x$$

When $x =_A y$ is inhabited (i.e., there exists $p : x =_A y$), we say that x and y are propositionally equal.

One can think of the identity type as the equality predicate in first order logic. But the identity type is best understood in the homotopy interpretation of type theory:

- Types are spaces,
- $a : A$ is a point in space A,
- $p : a =_A b$ is a path from a to b,
- $h : p =_{a =_A b} q$ is a “path”, or homotopy between the paths p and q.

Homotopical reading of identity types



$$p, q : x =_A y$$
$$H : p =_{x =_A y} q$$

The judgments of type theory

- The main judgment is $a : A$.

A type A is itself an element of a very large type, called $\mathcal{U} = \mathcal{U}_0$, for the universe. But $\mathcal{U}$ is also an element of an even larger type $\mathcal{U}_1$, etc., so there is a hierarchy of universes. (Setting $\mathcal{U} : \mathcal{U}$ would lead back to paradoxes!)

- Actually, judgments are in context, like for example

$$x : \mathbb{N} \vdash (2 \times x) : \mathbb{N}$$

A context is a list of type declarations. In the setting of dependent types, *the order of these declarations matters*, as types may depend on the previously declared variables, for example,

$$x : \mathbb{N}, y : \mathtt{Fin}(x)\ \mathtt{ctx}$$

(meaning that it is a valid context), while $y : \mathtt{Fin}(x), x : \mathbb{N}$ does not make sense.

- Finally, definitional (or judgmental) equality is also typed:

$$\Gamma \vdash a \equiv b : A$$

Informal conventions

In summary, there are three judgments:

$$\Gamma\ \mathtt{ctx} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash a \equiv b : A$$

We often omit Γ (or rather its "dummy" part that is not relevant for the discussion), and we also write $a : A$ for $\vdash a : A$.

We also often write A to mean that there exists $a : A$: think of A as a (provable) formula, and of a as a proof of A.

Some typing rules

Judgments are the base of the formal typing system of type theory. One derives judgments from other judgments. They are like phrases, while types or terms are more like nouns or verbs...

$$\frac{\Gamma \vdash A : \mathcal{U}}{\Gamma, x : A\ \mathtt{ctx}}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash A \equiv B : \mathcal{U}}{\Gamma \vdash a : B}$$

$$\frac{(i < j)}{\mathcal{U}_i : \mathcal{U}_j} \qquad \frac{A : \mathcal{U}_i}{A : \mathcal{U}_{i+1}}$$

Two kinds of equality

We have seen

- The definitional equality, for which the notation is $\equiv$. In these lectures, we also use the symbol $:\equiv$, which stresses more the definitional side, especially when defining macros.
- The propositional equality $a =_A b$.

Let us stress the difference:

- The definitional equality is set up at the level of judgments:

$$\Gamma \vdash a \equiv b : A$$

  This is why it is also called judgmental.

- The propositional equality is set up at the level of types:

$$x : A, y : A \vdash (x =_A y) : \mathcal{U}$$

Definitional equality is stronger, since if $a \equiv b$, then $(a =_A b) \equiv (a =_A a)$, and hence $\texttt{refl}(a) :\ a =_A b$.

How to prove equalities

• Definitional equality $a \equiv b : A$: Usual business of equational reasoning: go from a to b by a sequence of definitional equalities as provided by the various type formers (starting with the β-rules and the definitional equalities associated with induction operators, see below)

• Propositional equality $a =_A b$: Produce a term (or proof, or witness) $p : (a =_A b)$. Such a witness is often obtained by applying an adequate induction operator to an adequate type family of equality types (see the emblematic case of surjective pairing below).

Families of types

$\mathtt{Fin}(n)$ is a family of types, over $\mathtt{N}$. More precisely, we have

$$n : \mathbb{N} \vdash \mathtt{Fin}(n) : \mathcal{U}$$

We can also write

$$\mathtt{Fin} : \mathbb{N} \to \mathcal{U}$$

Successive dependencies: if $A : \mathcal{U}$ and $B : A \to \mathcal{U}$, the type of a family C depending on A and B is

$$C : \Pi_{x:A}\,(B(x) \to \mathcal{U})$$

We have:

$$x : A\,,\; y : Bx \vdash Cxy : \mathcal{U} \quad \text{and hence} \quad x : A\,,\; y : Bx\,,\; z : Cxy\ \mathtt{ctx}$$

Alternatively, we can type C as

$$C : \Pi_{\Sigma_{x:A}(B(x)}\mathcal{U}$$

(cf curryfication in functional programming:

$$A \to (B \to C) \quad \text{versus} \quad (A \times B) \to C$$

The Π-type

This is the dependent version of the function type.

- If $A : \mathcal{U}$ and $B : A \to \mathcal{U}$, then $\Pi_{x:A} B\,x : \mathcal{U}$.
- If $x : A \vdash b : B\,x$, then $\lambda x.b : \Pi_{x:A} B\,x$.
- if $a : A$ and $b : \Pi_{x:A} B$, then $b\,a : B\,a$
- We require the definitional equality $(\lambda x.b)a \equiv b[x \leftarrow a]$

For $A : \mathcal{U}$ and $B : \mathcal{U}$, we define

$$A \to B :\equiv \Pi_{x:A}(\lambda x.B)x \equiv \Pi_{x:A} B$$

The `swap` function

Let

- $A : \mathcal{U}$, $B : \mathcal{U}$, $C : A \to B \to \mathcal{U}$,
- $h : \Pi_{x:A}\Pi_{y:B}.Cxy$

We want

$$\texttt{swap}\, A\, B\, C\, h$$

to swap the arguments of h (which is possible, because neither B depends on A nor A depends on B).

We set

$$\texttt{swap} :\equiv \lambda A \lambda B \lambda C \lambda h \lambda y \lambda x . h\, x\, y : \\ \Pi_{A:\mathcal{U}}\Pi_{B:\mathcal{U}}\Pi_{C:A\to B\to\mathcal{U}}((\Pi_{x:A}\Pi_{y:B}\, C\, x\, y) \to (\Pi_{y:B}\Pi_{x:A}\, C\, x\, y))$$

Such a function is called polymorphic.

The coproduct type

- If $A : \mathcal{U}$ and $B : \mathcal{U}$, then $A + B : \mathcal{U}$.
- If $a : A$ (resp. $b : B$), then $\texttt{inl}\, a : A + B$ (resp. $\texttt{inr}\, b : A + B$).

Induction operator: We postulate

$$\texttt{ind}_{A+B} : \Pi_{C:(A+B)\to\mathcal{U}}(\Pi_{a:A} C\,(\texttt{inl}\, a)) \to (\Pi_{b:B} C\,(\texttt{inr}\, b)) \to \Pi_{z:A+B} C\, z$$

(intuition: every inhabitant of $A + B$ comes from A or B).

- We require the definitional equalities

$$\texttt{ind}_{A+B}\, C\, f\, g\,(\texttt{inl}\, a) \equiv f\, a \qquad \texttt{ind}_{A+B}\, C\, f\, g\,(\texttt{inr}\, b) \equiv g\, b$$

The type of booleans

It is the special case $1 + 1$. It can be introduced directly:

- $2 : \mathcal{U}$
- The constructors are $0_2 : 2$ and $1_2 : 2$.
- Induction operator:

$$\mathtt{ind}_2 : \Pi_{C:2\to\mathcal{U}}\,(C\,0_2) \to (C\,1_2) \to \Pi_{z:2} C\,z$$

(intuition: every boolean is either 0_2 or 1_2).

- We require the definitional equalities

$$\mathtt{ind}_2\,C\,c_0\,c_1\,0_2 \equiv c_0 \qquad \mathtt{ind}_2\,C\,c_0\,c_1\,1_2 \equiv c_1$$

We also spell out the recursion operator:

$$\mathtt{rec}_2 : \Pi_{C:\mathcal{U}}\,C \to C \to (2 \to C)$$

$$\mathtt{rec}_2\,C\,c_0\,c_1\,0_2 \equiv c_0 \qquad \mathtt{rec}_2\,C\,c_0\,c_1\,1_2 \equiv c_1$$

The Σ-type

This is the dependent version of the product type.

- If $A : \mathcal{U}$ and $B : A \to \mathcal{U}$, then $\Sigma_{x:A} B\, x : \mathcal{U}$.
- If $a : A$ and $b : B\, a$, then $(a, b) : \Sigma_{x:A} B\, x$.

  We bootstrap the induction operator:

  $$\begin{aligned} &\mathtt{rec}_{A\times B} : \Pi_{C:\mathcal{U}}(A \to B \to C) \to ((A \times B) \to C) \\ &\mathtt{ind}_{A\times B} : \Pi_{C:(A\times B)\to\mathcal{U}}(\Pi_{x:A,y:B} C\,(x, y)) \to \Pi_{z:A\times B} C\, z \\ &\mathtt{ind}_{\Sigma_{x:A} B\, x} : \Pi_{C:(\Sigma_{x:A} B\, x)\to\mathcal{U}}(\Pi_{x:A,y:B\, x} C\,(x, y)) \to \Pi_{z:\Sigma_{x:A} B\, x} C\, z \end{aligned}$$

  (intuition: every inhabitant of $\Sigma_{x:A} B\, x$ is a pair)
- We require the definitional equality $\mathtt{ind}_{\Sigma_{x:A} B}\, C\, f\, (a, b) \equiv f\, a\, b$.

For $A : \mathcal{U}$ and $B : \mathcal{U}$, we define

$$A \times B :\equiv \Sigma_{x:A}(\lambda x.B)x \equiv \Sigma_{x:A} B$$

We can complete the list by spelling out the non dependent version of $\mathtt{ind}_{\Sigma_{x:A} B\, x}$:

$$\mathtt{rec}_{\Sigma_{x:A} B\, x} : \Pi_{C:\mathcal{U}}(\Pi_{x:A}(Bx \to C)) \to ((\Sigma_{x:A} B\, x) \to C)$$

Projections associated with Σ-types

We set

$$\begin{array}{ll} \mathtt{pr}_1 : (\Sigma_{x:A} Bx) \to A & \mathtt{pr}_1 :\equiv \mathtt{ind}_{\Sigma_{x:A} Bx} \, (\lambda z.A) \, (\lambda x \lambda y.x) \\ \mathtt{pr}_2 : \Pi_{a:\Sigma_{x:A} Bx} B(\mathtt{pr}_1 a) & \mathtt{pr}_2 :\equiv \mathtt{ind}_{\Sigma_{x:A} Bx} \, (\lambda z.B \, (\mathtt{pr}_1 z)) \, (\lambda x \lambda y.y) \end{array}$$

Slowly, setting $C :\equiv \lambda z.B \, (\mathtt{pr}_1 z)$, $\mathtt{ind}_{\Sigma_{x:A} Bx} \, (\lambda z.B \, (\mathtt{pr}_1 z))$ expects an argument of type $\Pi_{x:A} \Pi_{y:Bx} C(x, y) \equiv \Pi_{x:A} \Pi_{y:Bx} Bx$, so that the definition of $\mathtt{pr}_2$ type-checks.

The following definitional equalities hold:

$$\mathtt{pr}_1(x, y) \equiv x \quad \mathtt{pr}_2(x, y) \equiv y$$

Another equality that we could hope for is the equality between $(\mathtt{pr}_1 z, \mathtt{pr}_2 z)$ and z (for $z : \Sigma_{x:A} Bx$): it holds, but only propositionally (surjective pairing) .

Surjective pairing

Let

$$C :\equiv \lambda z.\,((\mathtt{pr}_1 z, \mathtt{pr}_2 z) =_{\Sigma_{x:A} Bx} z)$$

We have, for $x : A$ and $y : Bx$, that $\mathtt{refl}((x,y)) : (x,y) =_{\Sigma_{x:A} Bx} (x,y)$.
But we have $(x,y) \equiv (\mathtt{pr}_1(x,y), \mathtt{pr}_2(x,y))$. Hence $\mathtt{refl}((x,y)) : C(x,y)$.
We set

$$\mathtt{surjpair} :\equiv \mathtt{ind}_{\Sigma_{x:A} Bx} C\,(\lambda x\,y.\mathtt{refl}((x,y))) : \Pi_{z:\Sigma_{x:A} Bx} Cz$$

We have:

$$\mathtt{surjpair}(z) : (\mathtt{pr}_1 z, \mathtt{pr}_2 z) =_{\Sigma_{x:A} Bx} z$$

The types 1 and 0

They are the 0-ary versions of the product and coproduct constructors, respectively.

- $1 : \mathcal{u}$, $\star : 1$, and the induction operator

$$\mathtt{ind}_1 : \Pi_{C:1 \to \mathcal{U}}(C\,\star) \to \Pi_{z:1}(C\,z)$$

- $0 : \mathcal{u}$, there is no constructor, and there is an induction operator

$$\mathtt{ind}_0 : \Pi_{D:0 \to \mathcal{U}} \Pi_{x:0}(D\,x)$$

.

As an illustration, we see that

$$x : 0, y : 0 \vdash \mathtt{ind}_0\, D\, x : x =_0 y$$

where $D\,x :\equiv (x =_0 y)$ (y fixed). With anticipation, this says that 0 is a proposition.

The type of natural numbers

- $\mathbb{N} : \mathcal{U}$
- The constructors are given by $0 : \mathbb{N}$ and $\mathtt{succ} : \mathbb{N} \to \mathbb{N}$
- Induction operator:

$$\mathtt{ind}_{\mathbb{N}} : \Pi_{C:\mathbb{N}\to\mathcal{U}} \; (C\,0) \to (\Pi_{n:\mathbb{N}}(C\,n \to C\,(\mathtt{succ}\,n))) \to \Pi_{n:\mathbb{N}} C\,z$$

We require the definitional equalities

$$\mathtt{ind}_{\mathbb{N}}\,C\,c_0\,c_s\,0 \equiv c_0 \qquad \mathtt{ind}_{\mathbb{N}}\,C\,c_0\,c_s\,(\mathtt{succ}\,n) \equiv c_s\,n\,(\mathtt{ind}_{\mathbb{N}}\,C\,c_0\,c_s\,n)$$

- Non dependent version (primitive recursion):

$$\mathtt{rec}_{\mathbb{N}} : \Pi_{C:\mathcal{U}} \; C \to (\mathbb{N} \to (C \to C)) \to (\mathbb{N} \to C)$$

- There is also the (seemingly) more rudimentary iteration:

$$\mathtt{iter} : \Pi_{C:\mathcal{U}} \; C \to (C \to C) \to (\mathbb{N} \to C)$$

with $\mathtt{iter}\,C\,c_0\,c_s\,(\mathtt{succ}\,n) \equiv c_s(\mathtt{iter}\,C\,c_0\,c_s\,n)$.

- There is also dependent iteration, which is to $\mathtt{iter}$ what $\mathtt{ind}_{\mathbb{N}}$ is to $\mathtt{rec}_{\mathbb{N}}$.

Defining *n*-ary functions by primitive recursion

Primitive recursive functions have typically n arguments. For $n = 2$, one would like to have (with $g : \mathbb{N} \to C$ and $h : (\mathbb{N} \times C \times \mathbb{N}) \to C$):

$$f :\equiv \ \mathtt{rec}_{\mathbb{N}\times\mathbb{N}}\, C\, g\, h : \ \mathbb{N} \times \mathbb{N} \to C$$

such that

$$\begin{aligned} f(0, m) &:\equiv g\, m \\ f(\mathtt{succ} n, m) &:\equiv h(n, f(n, m), m) \end{aligned}$$

This can be encoded by the unary version. The trick is to define $\mathtt{rec}_{\mathbb{N}\times\mathbb{N}}\, C\, g\, h$ as a function from $\mathbb{N}$ to $C' :\equiv \mathbb{N} \to C$:

$$\mathtt{rec}_{\mathbb{N}\times\mathbb{N}}\, C\, g\, h :\equiv \ \mathtt{rec}_{\mathbb{N}}\, C'\, g\, (\lambda n \lambda f \lambda m. h(n, fm, m))$$

Encoding recursion from iteration (propositionally)

We have, for $c_0' : D$ and $c_s' : D \to D$: $\texttt{iter}\, D\, c_0'\, c_s' : \mathbb{N} \to D$. The trick to encode $\texttt{rec}_{\mathbb{N}}\, c_0\, c_s$, with $c_0 : C$ and $c_s : \mathbb{N} \to C \to C$, is to gather all the functions $(c_s\, n)$ into a single function. So we set:

$$D :\equiv \mathbb{N} \times C \quad c_0' :\equiv (0, c_O) \quad c_s'(n, c) :\equiv (\texttt{succ}(n), c_s\, n\, c)$$

We define $\texttt{rec}_{\mathbb{N}}\, C\, c_0\, c_s\, n :\equiv \texttt{pr}_2(\texttt{iter}\, D\, c_0'\, c_s'\, n)$.

- One checks easily that $\texttt{rec}_C\, c_0\, c_s\, 0 \equiv c_0$.
- But the other equation holds only propositionally. One establishes:

$$\begin{array}{ll} \texttt{pr}_1(c_s' z) = \texttt{succ}(\texttt{pr}_1 z) & \text{(product induction)} \\ \texttt{pr}_1(\texttt{iter}\, D\, c_0'\, c_s'\, n) = n & \text{(non dependent iteration)} \end{array}$$

We set $a :\equiv \texttt{iter}\, D\, c_0'\, c_s'\, n$. We get (using surjective pairing):

$$\begin{array}{rcl} \texttt{rec}_{\mathbb{N}}\, C\, c_0\, c_s\, (\texttt{succ}(n)) & \equiv & \texttt{pr}_2(c_s' a) \\ & =_C & \texttt{pr}_2(c_s'(n, \texttt{pr}_2 a)) \\ & \equiv & \texttt{pr}_2(\texttt{succ}(n), c_s\ n\, (\texttt{pr}_2 a)) \\ & \equiv & c_s\ n\, (\texttt{pr}_2 a) \\ & \equiv & c_s(n, \texttt{rec}_{\mathbb{N}}\, C\, c_0\, c_s\, n)) \end{array}$$

The identity type

- If $A : \mathcal{U}$, then $x : A, y : A \vdash (x =_A y) : \mathcal{U}$.
- Constructor: if $a : A$, then $(\mathtt{refl}(a)) : (a =_A a)$.
- Induction operator (path induction):

$$\mathtt{ind}_{=_A} : \Pi_{C : \Pi_{x,y:A}((x =_A y) \to \mathcal{U})} \left(\Pi_{x:A} C\, x\, x\, \mathtt{refl}(x)\right) \to \left(\Pi_{x,y:A} \Pi_{p : x =_A y} C\, x\, y\, p\right)$$

(intuition: the only *generic* element of the identity type is $\mathtt{refl}$).

We require the definitional equality

$$\mathtt{ind}_{=_A}\, C\, c\, x\, x\, (\mathtt{refl}(x)) \equiv c\, x$$

Path induction (slowly)

• In English, given a family of types $C\,x\,y\,p$, in order to define a dependent function in $\Pi_{x,y:A}\Pi_{p:x=_A y}C\,x\,y\,p$, all we need to have is its restriction to inputs of the form $(x\,,\,x\,,\mathtt{refl}(x))$.
Here it is important to have both x, y variables. Think of x and y sliding toward each other along p.

• This game of sliding also works if x is fixed (think then of y sliding all the way to x along p), giving rise to the following other induction principle (based path induction):

$$\mathtt{ind}'_{=_A} : \Pi_{x:A}\Pi_{D:\Pi_{y:A}(x=_A y)\to\mathcal{U}}\; D\,x\,(\mathtt{refl}(x)) \to \Pi_{y:A}\Pi_{p:x=_A y}D\,y\,p$$

Definitional equality: $\mathtt{ind}'_{=_A}\,a\,D\,u\,a\,(\mathtt{refl}(a)) \equiv u$.
In English, suppose $a : A$ in A is fixed, and that a family of types $D\,y\,p$ $(p : a =_A y)$ is given. In order to define a dependent function in $\Pi_{y:A}\Pi_{p:x=_A y}D\,y\,p$, all we need to have is its value u at $(a\,,\,a\,,\mathtt{refl}(a))$.

• If x and y are fixed, then no sliding is possible! At least one must be variable.

The interdefinability of $\mathtt{ind}_{=_A}$ and $\mathtt{ind}'_{=_A}$

$$\begin{aligned}\mathtt{ind}_{=_A} &: \Pi_{C:\Pi_{x,y:A}(x=_A y)\to\mathcal{U}}\Pi_{x:A}\, C\,x\,x\,\mathtt{refl}(x) \to \Pi_{x,y:A}\Pi_{p:x=_A y} C\,x\,y\,p\\ \mathtt{ind}'_{=_A} &: \Pi_{x:A}\Pi_{D:\Pi_{y:A}(x=_A y)\to\mathcal{U}}\, D\,x\,(\mathtt{refl}(x)) \to \Pi_{y:A}\Pi_{p:x=_A y} D\,y\,p\end{aligned}$$

One defines $\mathtt{ind}$ from $\mathtt{ind}'$ by instantiation:

$$\mathtt{ind}_{=_A}\, C\,c\,x\,y\,p :\equiv \mathtt{ind}'_{=_A}\, x\,(C\,x)\,(c\,x)\,y\,p$$

Conversely, in order to define $\mathtt{ind}'_{=_A}\, a\,D\,d\,y\,p$, one uses a polymorphic trick (due to Christine Paulin) to get an appropriate instantiation of $\mathtt{ind}$. We set

$$C :\equiv \lambda x\lambda y\lambda p.\Pi_{E:\Pi_{z:A}(x=_A z)\to\mathcal{U}} E\,x\,(\mathtt{refl}(x)) \to E\,y\,p$$

and then

$$\mathtt{ind}'_{=_A}\, a\,D\,d\,y\,p :\equiv \mathtt{ind}_{=_A}\, C\,(\lambda x\lambda E\lambda e.\,e)\,a\,y\,p\,D\,d$$

The general pattern for type constructors

- Formation rules for the new type.
- Introduction rules for the new type, featuring the new constructors:

$$\lambda x.a \quad (a, b) \quad \mathtt{inl}(a)\ \mathtt{inr}(a) \quad 0_2 \quad 1_2 \quad \mathtt{refl}(x)$$

- Rules specifying how the type is used: application, induction operators (when the induction operator is applied to a constant type family, we call it recursion operator).
- Associated definitional (or propositional) equalities.
- Sometimes a uniqueness principle holds (or is assumed) propositionally: for example surjective pairing, or $\lambda x.ax = a$.

Induction principles in a nutshell

$$\mathtt{ind}_{\Sigma_{x:A} B\,x} : \Pi_{C:(\Sigma_{x:A} B\,x)\to\mathcal{U}} (\Pi_{x:A,y:B\,x} C\,(x,y)) \to \Pi_{z:\Sigma_{x:A} B\,x} C\,z$$

$$\mathtt{ind}_{A+B} : \Pi_{C:(A+B)\to\mathcal{U}} (\Pi_{a:A} C\,(\mathtt{inl}\,a)) \to (\Pi_{b:B} C\,(\mathtt{inr}, b)) \to \Pi_{z:A+B} C\,z$$

$$\mathtt{ind}_{2} : \Pi_{C:2\to\mathcal{U}} (C\,0_1) \to (C\,1_2) \to \Pi_{z:2} C\,z$$

$$\mathtt{ind}_{\mathbb{N}} : \Pi_{C:\mathbb{N}\to\mathcal{U}} (C\,0) \to (\Pi_{n:\mathbb{N}} (\mathbb{N} \to (C\,n) \to (C\,(\mathtt{succ}\,n)))) \to \Pi_{n:\mathbb{N}} C\,z$$

$$\mathtt{ind}_{=_A} : \Pi_{C:\Pi_{x,y:A}(x=_A y)\to\mathcal{U}} \Pi_{x:A}\ C\,x\,x\,\mathtt{refl}(x) \to \Pi_{x,y:A} \Pi_{p:x=_A y} C\,x\,y\,p$$
$$\mathtt{ind}'_{=_A} : \Pi_{x:A} \Pi_{D:\Pi_{y:A}(x=_A y)\to\mathcal{U}}\ D\,x\,(\mathtt{refl}(x)) \to \Pi_{y:A} \Pi_{p:x=_A y} D\,y\,p$$

Elementary properties involving identity types

The following properties are easy applications of path induction:

• Paths can be composed (or concatenated) (notation $p \cdot q$) and inverted (notation p^{-1}).

• These operations have the structure of a weak groupoid, with `refl` as identity:

$$\mathtt{refl}{\cdot}p = p = p{\cdot}\mathtt{refl} \quad (p{\cdot}q){\cdot}r = p{\cdot}(q{\cdot}r) \quad p{\cdot}p^{-1} = \mathtt{refl} \quad p^{-1}{\cdot}p = \mathtt{refl}$$

• Functions are functors: if $A : \mathcal{U}, B : \mathcal{U}$, if $f : A \to B$, and if $x, y : A$ and $p : (x =_A y)$, then we can define

$$\mathsf{ap}\, f\, x\, y\, p : ((f\, x) =_B (f\, y)) \qquad \text{(abbreviated as } (\mathsf{ap}\, f\, p) \text{ or } f(p))$$

by path induction, setting $\mathsf{ap}\, f\, x\, x\, (\mathtt{refl}(x)) :\equiv \mathtt{refl}(f\, x)$.

But what if $f : \Pi_{x:A}(B\, x)$ is a dependent function? We have that $(f\, x) : (B\, x)$ and $(f\, y) : (B\, y)$ do not live in the same type... So we must do something first.

Composition, inversion, group laws, slowly

Concatenation: For $p : x =_A y$ and $q : y =_A z$, to define $p \cdot q$, we can apply path induction to p, so that we are left to define $\mathtt{refl}(x) \cdot q$ for $q : x =_A z$. We then apply induction on q, so that we are left to define $\mathtt{refl}(x) \cdot \mathtt{refl}(x) :\equiv \mathtt{refl}(x)$.

Inversion: By path induction on p, we are left to define $(\mathtt{refl}(x))^{-1} :\equiv \mathtt{refl}(x)$.

Group laws: Again, by path induction! For example, to prove $p \cdot \mathtt{refl} = p$, it is enough to exhibit an inhabitant of $(\mathtt{refl} \cdot \mathtt{refl} = \mathtt{refl})$: take $\mathtt{refl}(\mathtt{refl})$.

Remark on composition We could have defined right away $\mathtt{refl} \cdot q :\equiv q$ (rather than starting a new induction on q), and it works, but it is not symmetric in p, q in that one can only prove $p \cdot \mathtt{refl} = p$ (propositional, not definitional). In summary, we can define composition in three ways, leading to propositionally equal definitions satisfying differents definitional equalities:

$$\mathtt{refl}(x) \cdot \mathtt{refl}(x) \equiv \mathtt{refl}(x) \quad \text{or} \quad \mathtt{refl} \cdot q :\equiv q \quad \text{or} \quad p \cdot \mathtt{refl} \equiv p$$

Transport

This is probably the most ubiquitous application of path induction. Consider a type family $C : A \to \mathcal{U}$. We can define the type family

$$D\,x\,y\,p :\equiv (C\,x) \to (C\,y)$$

Then

$$\mathtt{transport}^C :\equiv (\mathtt{ind}_{=_A}\,D\,(\lambda x.\mathtt{id}_{(C\,x)})) : \Pi_{x,y:A}\Pi_{p:(x=_A y)}(C\,x) \to (C\,y)$$

We shall write $(\mathtt{transport}^C\,p)$ for $(\mathtt{transport}^C\,x\,y\,p)$.

In English, given $p : (x =_A y)$ and $u : (C\,x)$, $(\mathtt{transport}\,p\,u)$ **transports** u from $(C\,x)$ to $(C\,y)$.

We can then define, for $f : \Pi_{x:A}(C\,x)$, $x, y : A$ and $p : (x =_A y)$:

$$\mathtt{apd}\,f\,x\,y\,p : ((\mathtt{transport}\,p\,(f\,x)) =_{B y} (f\,y))$$

by $\mathtt{apd}\,f\,x\,x\,\mathtt{refl}(x) :\equiv \mathtt{refl}(f\,x)$.

Equivalence of types

We set (pointwise equality)

$$f \sim_{A\to B} g :\equiv \Pi_{x:A}(f\,x =_B g\,x)$$

An inhabitant of $f \sim_{A\to B} g$ is called a homotopy from f to g.
We say that A, B are equivalent via $f : A \to B$ if there is an inhabitant in $\mathtt{qinv}(f)$, where

$$\mathtt{qinv}(f) :\equiv \Sigma_{g:B\to A}\,(f \circ g \sim \mathtt{id}) \times (g \circ f \sim \mathtt{id})$$

The function $g : B \to A$ is called a quasi-inverse of f. We use the (personal) notation $A \overset{\sim}{\longleftrightarrow} B$ for this.

We shall also consider a weaker notion: two types A and B are logically equivalent (notation $A \longleftrightarrow B$) if we can exhibit $f : A \to B$ (written a $A \longrightarrow B$) and $g : B \to A$.

Some properties of transport

• We have, by definition, for $C : A \to \mathcal{U}$, $x : A$ and $u : (C\,x)$:

$$(\mathtt{transport}^C\,(\mathtt{refl}(x))\,u) \equiv u$$

One shows by path induction that transport respects composition of paths:

$$(\mathtt{transport}^C\,q) \circ (\mathtt{transport}^C\,p) \sim (\mathtt{transport}^C\,(p \cdot q))$$

• If $C : B \to \mathcal{U}$, $f : A \to B$, $p : (a_1 =_A a_2)$ and $u : (C\,(f\,a_1))$, then

$$\mathtt{transport}^C\,(f(p))\,u = \mathtt{tansport}^{C \circ f}\,p\,u$$

• If $A : \mathcal{U}$ and $C : A \to \mathcal{U}$ is defined by $C\,x :\equiv (a_1 =_A x)$ (a_1 fixed), then, for $p : (a_2 =_A a_3)$ and $q : (C\,a_2)$, we have:

$$\mathtt{transport}^C\,p\,q = q \cdot p$$

ap versus apd

If we apply $\mathtt{apd}$ for a constant family $\lambda x : A.B$, then, for $f : A \to B$ and $p : x =_A y$, $\mathtt{apd}\, f\, p$ and $\mathtt{ap}\, f\, p$ do not have the same type:

$$\mathtt{ap}\, f\, p : (fx =_B fy) \qquad \mathtt{apd}\, f\, x : ((\mathtt{transport}^{\lambda x:A.B}\, p\, (fx)) =_B fy)$$

But they are related.

• For all $b : B$ $x, y : A$ and $p : x =_A y$, there exists a path

$$\mathtt{transportconst}_p^{\lambda x:A.B}(b) : (\mathtt{transport}^{\lambda x:A.B}\, p\, b) =_B b$$

This is clear by path induction: since $\mathtt{transport}^{\lambda x:A.B}\, \mathtt{refl}(x)\, b \equiv b$, we can choose $\mathtt{refl}(b)$.

• we have $\mathtt{apd}\, f\, p = \mathtt{transportconst}_p^{\lambda x:A.B}(fx) \cdot \mathtt{ap}\, f\, p$.
We prove this by path induction. We are reduced to prove

$$\mathtt{apd}\, f\, \mathtt{refl}(x) = \mathtt{transportconst}_{\mathtt{refl}(x)}^{\lambda x:A.B}(fx) \cdot \mathtt{ap}\, f\, \mathtt{refl}(x)$$

which by definition is exactly

$$\mathtt{refl}(f(x)) = (\mathtt{refl}(f(x)) \cdot \mathtt{refl}(f(x))) \equiv \mathtt{refl}(f(x)),$$

so that we can choose $\mathtt{refl}(\mathtt{refl}(f(x)))$ as witness.

Some properties of the types $f \sim g$

•. Homotopy is an equivalence relation. We show transitivity. Let $H : (f \sim g)$ and $K : (g \sim h)$. Then

$$\lambda x.((H\,x) \cdot (K\,x)) : (f \sim h) \quad \text{(using path concatenation)}$$

• If $H : (f \sim_{A \to B} g)$ and $k : A' \to A$. Then

$$H \circ k :\equiv \lambda x'.H(k\,x') : (f \circ k \sim_{A' \to B} g \circ k)$$

• If $H : (f \sim_{A \to B} g)$ and $h : B \to B'$. Then

$$\lambda x.h(Hx) : \ (h \circ f \sim_{A \to B'} h \circ g)$$

An example of logical equivalence

We set

$$\mathtt{biinv}(f) :\equiv \left(\Sigma_{g:B\to A}(f\circ g \sim \mathtt{id})\right) \times \left(\Sigma_{h:B\to A}(h\circ f \sim \mathtt{id})\right)$$

We have

$$\mathtt{qinv}(f) \longleftrightarrow \mathtt{biinv}(f)$$

Proof: In one direction, if g is a quasi-inverse, then take g, h to be both g. Conversely, we note that if $f\circ g \sim \mathtt{id}$ and $h \circ f \sim \mathtt{id}$, then (using the properties listed in the previous slide)

$$g \sim (h\circ f)\circ g \sim h\circ(f\circ g) \sim h,$$

and hence $g\circ f \sim h\circ f \sim \mathtt{id}$, establishing $(\mathtt{qinv}\, f)$ via g.

Characterising $=_{A \times B}$

We have:

$$(x =_{A \times B} y) \xleftrightarrow{\sim} ((\mathtt{pr}_1\, x =_A \mathtt{pr}_1\, y) \times (\mathtt{pr}_2\, x =_B \mathtt{pr}_2\, y))$$

We have

$$\lambda p.(\mathtt{pr}_1(p), (\mathtt{pr}_2(p)) : (x =_{A \times B} y) \to (\mathtt{pr}_1\, x =_A \mathtt{pr}_1\, y) \times (\mathtt{pr}_2\, x =_B \mathtt{pr}_2\, y)$$

For defining a map $\mathtt{pair}^=$ in the other direction, we use induction for cartesian products at two levels:

- assume that $x \equiv (a, b)$ and $y \equiv (a', b')$,
- assume that we have $p : (a =_A a')$ and $q : (b =_B b')$,

and then path induction: we can assume p, q to be `refl`, and set

$$(\mathtt{pair}^=)\,((\mathtt{refl}(a)), (\mathtt{refl}(b))) :\equiv \mathtt{refl}(a, b)$$

And one needs to show that $\mathtt{pair}^=$ is a quasi inverse of $\lambda p.((\mathrm{ap}\, \mathtt{pr}_1\, p), (\mathrm{ap}\, \mathtt{pr}_2\, p))$: do path inductions in the right order!

Characterising $=_{\Sigma_{x:A} B\,x}$

We have similarly an equivalence between $(w =_{\Sigma_{x:A}(B\,x)} w')$ and

$$\Sigma_{p:(\mathtt{pr}_1\,w) =_A (\mathtt{pr}_1 w')}((\mathtt{transport}\,p\,(\mathtt{pr}_2\,w)) =_{B\,(\mathtt{pr}_1\,w')} (\mathtt{pr}_2\,w'))$$

We write again $\mathtt{pair}^{=}$ for the quasi-inverse.

Characterising $=_1$

We have

$$(x =_1 y) \overset{\sim}{\longleftrightarrow} 1$$

Let $x : 1, y : 1$. We have $\lambda p.\star : (x =_1 y) \to 1$. In the converse direction, let $B : 1 \to \mathcal{U}$ be the constant family defined by $(B\, z) :\equiv (x =_1 y)$. ($x, y$ fixed). Then, by induction on 1, all we need is an inhabitant $\Box_1 : (x =_1 y)$.

For this, we need to be generic in x, y. We shall exhibit an inhabitant $\Box_2 : \Pi_{x,y:1}(x =_1 y)$.

We set $(C\, x) :\equiv \Pi_{y:1}(x =_1 y)$. By induction on 1, all we need is an inhabitant $\Box_3 : (C\, \star) \equiv \Pi_{y:1}(\star =_1 y)$.

We set $D\, y :\equiv (\star =_1 y)$. By induction on 1, we can set

$$\begin{aligned}
\Box_3 &:\equiv (\mathtt{ind}_1\, D\, (\mathtt{refl}\, \star)) \\
\Box_2 &:\equiv (\mathtt{ind}_1\, C\, \Box_3) \\
\Box_1 &:\equiv (\Box_2\, x\, y)
\end{aligned}$$

Finally $(\mathtt{ind}_1\, B\, \Box_1) : 1 \to (x =_1 y)$ is the desired inverse.

Function extensionality and univalence

We would like to characterise the equality for function types (and Π-types) and for universes. For this, we need additional **axioms**, which are:

- Function extensionality: We assume

$$(f =_{A \to B} g) \overset{\sim}{\longleftrightarrow} (f \sim_{A \to B} g)$$

- Univalence: We set

$$A \cong B :\equiv \Sigma_{f : A \to B} \, \mathtt{biinv}(f)$$

and write witnesses of this type as $(f, g, h, \epsilon, \eta)$. Then we assume

$$(A =_{\mathcal{U}} B) \overset{\sim}{\longleftrightarrow} (A \cong B)$$

where in both cases the equivalence is via the canonical morphism from the equality type to its characterisation.

Note that we used `biinv` and not `qinv` in the definition of $A \cong B$. More on this later.

Canonical morphisms (function extensionality, univalence)

• The family $\mathtt{happly}(f, g)$ of canonical morphisms from $(f =_{A\to B} g)$ to $(f \sim_{A\to B} g)$ is defined by path induction: when f and g coincide, then we set

$$\mathtt{happly}(f, f)(\mathtt{refl}(f)) :\equiv \lambda x.\mathtt{refl}(f(x)) : (f \sim_{A\to B} f)$$

• The canonical morphism $\mathtt{idtoeqv} : (A =_{\mathcal{U}} B) \to (A \cong B)$ is defined using transport for the family $\mathtt{id}_{\mathcal{U}} : \mathcal{U} \to \mathcal{U}$. Let $p : (A =_{\mathcal{U}} B)$. We have

$$\begin{array}{l} \mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, p : A \to B \\ \mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, p^{-1} : B \to A \\ (\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, p) \circ (\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, p^{-1}) \sim (\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, \mathtt{refl}) \\ (\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, p^{-1}) \circ (\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, p) \sim (\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, \mathtt{refl}) \end{array}$$

Therefore we can set $\mathtt{idtoeqv}\, p$ to be

$$((\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, p), (\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, p^{-1}), (\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, p^{-1}), \epsilon, \eta)$$

where ϵ, η witness the above two pointwise equalities (recall that $(\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, \mathtt{refl}) = \mathtt{id}$).

Zoom on `idtoeqv`

Slowly, we establish

$$(\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, p) \circ (\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, p^{-1}) \sim (\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, \mathtt{refl})$$

in two steps:

- We have already established

$$(\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, p) \circ (\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, p^{-1}) \sim (\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\, (p \cdot p^{-1}))$$

- We show that if $p = q$, then

$$(\mathtt{transport}\, p) \sim (\mathtt{transport}\, q)$$

  We fix p, x and define

$$C\; p\; q\; H :\equiv (\mathtt{transport}\, p\, x) = (\mathtt{transport}\, q\, x)$$

  By path induction, it is enough to find an element of $(C\; p\; p\, \mathtt{refl})$: take $\mathtt{refl}(\mathtt{transport}\, p\, x)$.

Two properties of ua

We recall that by definition of idtoeqv, we have, for $p : A =_{\mathcal{U}} B$:

$$\begin{aligned}&\texttt{transport}^{\texttt{id}_{\mathcal{U}}}\, p \equiv \texttt{pr}_1(\texttt{idtoeqv}\, p)\\ &\texttt{transport}^{\texttt{id}_{\mathcal{U}}}\, p^{-1} \equiv \texttt{pr}_2(\texttt{idtoeqv}\, p) \equiv \texttt{pr}_3(\texttt{idtoeqv}\, p)\end{aligned}$$

where $\texttt{pr}_1$ and $\texttt{pr}_2$ retrieve the two quasi-inverse fonctions (the second one being repeated to get a term of biinv type). If follows that for all $e \equiv (f, g, h, \epsilon, \eta) : A \cong B$ we have

$$\begin{aligned}&\texttt{transport}^{\texttt{id}_{\mathcal{U}}}\, (\texttt{ua}\, e) = f\\ &\texttt{transport}^{\texttt{id}_{\mathcal{U}}}(\texttt{ua}\, e)^{-1} = g = h\end{aligned}$$

Slowly, we have, e.g.:

$$\texttt{transport}^{\texttt{id}_{\mathcal{U}}}\, (\texttt{ua}\, e) = \texttt{pr}_1(\texttt{idtoeqv}\, (\texttt{ua}\, e)) = (\texttt{pr}_1\, e) = f$$

An application of univalence

In usual mathematics, if f is a bijection from A to B, and if we have a semigroup structure on A, with multiplication $m : A \times A \to A$, we can define a binary operation on B, by setting

$$m'(b_1, b_2) = f(m(f^{-1}(b_1), f^{-1}(b_2)))$$

We can then prove that m' is associative, and hence that we have defined a semigroup structure on B.

In **univalent** mathematics, the perspective is different. We define

$$\texttt{Semigroup}\, A :\equiv \Sigma_{m:(A\times A\to A)}\Pi_{x,y:A}(m(m(x,y),z) =_A m(x, m(y,z)))$$

- If $e : (A \cong B)$ and $(m, a) : (\texttt{Semigroup}\, A)$, then

$$\texttt{transport}^{\texttt{Semigroup}}(\texttt{ua}\, e)\,(m, a)$$

  is automatically a semigroup structure on B.
- We then need to do a bit of computation to unroll this structure, and to discover that it is indeed equal to some (m', a') where m' is the one constructed above.

Computing m'

We stress from the previous slide that rather than cooking up a definition of m', we derive its definition from general principles, and then compute it. We define, for $e \equiv (f, g, h, \epsilon, \eta)$:

$$m' :\equiv \mathtt{pr}_1\,(\mathtt{transport}^{\mathtt{Semigroup}}(\mathtt{ua}\,e)\,(m, a))$$

We have (with $R :\equiv \lambda X.((X \times X) \to X$ and $Q :\equiv \lambda X.(X \times X))$:

$$\begin{aligned}
& m'(b_1, b_2) \\
&\quad = (\mathtt{transport}^{R}\,(\mathtt{ua}\,e)\,m)\,(b_1, b_2) \quad \text{(using } (\|) \text{ below)} \\
&\quad = \mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}(\mathtt{ua}\,e)\,m\,(\mathtt{transport}^{Q}\,(\mathtt{ua}\,e)^{-1}\,(b_1, b_2)) \\
&\quad = (f\ m\,(\mathtt{transport}^{Q}\,(\mathtt{ua}\,e)^{-1}\,(b_1, b_2))) \\
&\quad = (f\ m\,((\mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\,(\mathtt{ua}\,e)^{-1}\,b_1), \mathtt{transport}^{\mathtt{id}_{\mathcal{U}}}\,(\mathtt{ua}\,e)^{-1}\,b_2))) \\
&\quad = (f\ (m\,((g\ b_1), (g\ b_2))))
\end{aligned}$$

where we have used the following property: of $Cx :\equiv \Sigma_{y:C_1x}\, C_2xy$, $x_{,}, x_2 : A$, $p : x_1 =_A x_2$, $a : C_1x_1$ and $b : C_2x_1a$ then

$$(\|) \quad \mathtt{pr}_1(\mathtt{transport}^{C}\,p\,(a, b) = \mathtt{transport}^{C_1}\,p\,a$$

We also used the non-dependent version of Exercise 7.

Homotopy fibers

• Let $f : X \to Y$. We define $\mathtt{fib}_f(y) :\equiv \Sigma_{x:X}\,(f(x) =_Y y)$ and $\mathtt{fib}_f :\equiv \lambda y.\mathtt{fib}_f(y) : Y \to \mathcal{U}$.

• Conversely, we can associate (Grothendieck construction) with an arbitrary $P : Y \to \mathcal{U}$ the function $\mathtt{pr}_1 : \Sigma_{y:Y}\, Py \to Y$.

(†) The following holds: for all $Y : \mathcal{U}$, $P : Y \to \mathcal{U}$ and $b : Y$ we have

$$\mathtt{fib}_{\mathtt{pr}_1}(b) \overset{\sim}{\longleftrightarrow} Pb$$

Proof: In one direction, for $r : Pb$, we set $\phi(r) :\equiv ((b, r), \mathtt{refl}(b))$. For the other direction, by induction on Σ-types, it is enough to define $\psi((y, r), q)$ for all $y : Y$, $r : Py$, and $q : y =_Y b$, and in turn, by based path induction, it is enough to set $\psi((b, r), \mathtt{refl}(b)) :\equiv r$. We have thus $\psi(\phi(r)) \equiv r$ and hence a fortiori $\psi \circ \phi \sim \mathtt{id}$. For the other direction, we use again based path induction: we have that

$$\mathtt{refl}(((b, r), \mathtt{refl}(b))) : \phi(\psi((b, r), \mathtt{refl}(b))) = ((b, r), \mathtt{refl}(b))$$

since $\phi(\psi((b, r), \mathtt{refl}(b))) \equiv ((b, r), \mathtt{refl}(b)))$.

Grothendieck equivalence

For $Y : \mathcal{U}$, we define $\mathcal{U}/Y = \Sigma_{X:\mathcal{U}}(X \to Y)$. We have

$$(Y \to \mathcal{U}) \overset{\sim}{\longleftrightarrow} \mathcal{U}/Y$$

We define (as anticipated):

$$\begin{aligned} \psi(P) &:\equiv (\Sigma_{y:Y} Py, \mathtt{pr}_1) \\ \phi(X, f) &:\equiv \mathtt{fib}_f \end{aligned}$$

We have $\phi(\psi(P)) \equiv \mathtt{fib}_{\mathtt{pr}_1}$, and we deduce $\phi(\psi(P)) = P$ by univalence and by function extensionality from (†).

For the other direction, we need a charaterization of $=_{\mathcal{U}/Y}$, which itself relies on the following property:

For $p : A =_{\mathcal{U}} A'$, $f : A \to Y$, $f' : A' \to Y$, we have

$$(\mathtt{transport}^{\lambda Z.Z \to Y}\, p\, f =_{A' \to Y} f') \overset{\sim}{\longleftrightarrow} (f \sim (f' \circ \mathtt{pr}_1(\mathtt{idtoeqv}(p))))$$

Proof: By path induction, it is enough to check this for $\mathtt{refl}(A)$, and it is clear since $(\mathtt{transport}^{\lambda Z.Z \to Y}\, \mathtt{refl}(A)\, f) \equiv f$ and $\mathtt{pr}_1(\mathtt{idtoeqv}(\mathtt{refl}(A))) \equiv \mathtt{id}_A$.

Characterising the equality in $\mathcal{U}/Y$

(‡) Let $A, A', Y : \mathcal{U}$, $f : A \to Y$, $f' : A' \to Y$. Then we have

$$((A, f) =_{\mathcal{U}/Y} (A', f')) \overset{\sim}{\longleftrightarrow} \Sigma_{e:A \cong A'} (f \sim f' \circ \mathtt{pr}_1(e))$$

Proof: We have

$$\begin{aligned}
&((A, f) =_{\mathcal{U}/Y} (A', f')) \\
&\quad \overset{\sim}{\longleftrightarrow} \Sigma_{p:A =_{\mathcal{U}} A'}(\mathtt{transport}^{\lambda Z.Z \to Y}\, p\, f =_{A' \to Y} f') && \text{(induction on Σ-types)} \\
&\quad \overset{\sim}{\longleftrightarrow} \Sigma_{p:A =_{\mathcal{U}} A'} (f \sim f' \circ \mathtt{pr}_1(\mathtt{idtoeqv}(p))) && \text{(just proved above)} \\
&\quad \overset{\sim}{\longleftrightarrow} \Sigma_{e:A \cong A'} (f \sim f' \circ \mathtt{pr}_1(e)) && \text{(by univalence)}
\end{aligned}$$

In other words, proving $(A, f) =_{\mathcal{U}/Y} (A', f')$ consists in finding $\epsilon : A \to A'$ such that

$$\mathtt{biinv}(\epsilon) \quad \text{and} \quad f \sim f' \circ \epsilon$$

Back to Grothendieck equivalence

We have $\psi(\phi(X, f)) \equiv (\Sigma_{y:Y}\mathtt{fib}_f(y), \mathtt{pr}_1)$, hence by (‡) we want

$$\epsilon : X \to \Sigma_{y:Y}\mathtt{fib}_f(y) \quad \text{such that} \quad \mathtt{biinv}(\epsilon) \quad \text{and} \quad \mathtt{pr}_1 \circ \epsilon \sim f$$

We take

$$\epsilon(x) :\equiv (f(x), (x, \mathtt{refl}(f(x))))$$

Since $\mathtt{pr}_1 \circ \epsilon \equiv f$, we get immediately $\mathtt{pr}_1 \circ \epsilon \sim f$. We shall prove $\mathtt{qinv}(\epsilon)$, which a fortiori implies $\mathtt{biinv}(\epsilon)$. We define $\alpha : \Sigma_{y:Y}\mathtt{fib}_f(y) \to X$ by

$$\alpha(y, (x, r)) :\equiv x$$

We have $\alpha(\epsilon(x)) \equiv x$ by definition. For the converse direction, we use induction on Σ-types. We have to prove, for all x, y, and $r : (f(x) = y)$:

$$(y, (x, r)) = (f(x), (x, \mathtt{refl}(f(x)))) \equiv \epsilon(\alpha(y, x, r))$$

We can leave x (and hence $f(x)$) fixed and conclude by based path induction.

Reflecting on Grothendieck equivalence

The theorem just proved says that any pair of (Z, f) of a type Z and a map $f : Z \to X$ is equal to a pair of the form $(\Sigma_{x:X} Px, \mathtt{pr}_1)$, for a family $P : X \to \mathcal{U}$ of types depending on X.

Now, for an arbitrary $P : X \to \mathcal{U}$, the pair $(\Sigma_{x:X} Px, \mathtt{pr}_1)$ enjoys a remarkable property (typical of fibrations in topology and category theory):

(\$) For any $a, b : X$ $p : a =_X b$ and $u : Xa$, we have a path

$$q : (a, u) =_{\Sigma_{x:X} Px} (b, (\mathtt{transport}^P\, p\, u))$$

such that $\mathtt{pr}_1(q) :\equiv p$.
The proof of (\$) is obvious: take $(p, \mathtt{refl}(\mathtt{transport}^P\, p\, u))$.

Thus, loosely speaking, by combining this remark with the Grothendieck equivalence, we have that every map in type theory has a fibration-like flavour.

These considerations justify the terminology of calling $\Sigma_{x:X} Px$ the total space of the family P.

Mere propositions, and contractible types

A type A is a (mere) proposition if we can exhibit an inhabitant of

$$\Pi_{x,y:A}(x = y)$$

A type A is contractible if we can exhibit an inhabitant of

$$\Sigma_{x:A}\Pi_{y:A}\,(x = y),$$

i.e.:

- an inhabitant a of A (called center of contraction),
- and an inhabitant of $\Pi_{y:A}\,(a = y)$.

"Center" is a way of speaking, as in fact, any inhabitant of a contractible type may serve as center.

An example of contractible type (solution to Exercise 11)

For any type A and $a : A$, the type $\Sigma_{x:A}(a =_A x)$ is contractible.

Proof: A candidate for being a center is $(a, \mathtt{refl}(a))$. For any $(x, p) : \Sigma_{x:A}(a =_A x)$ (thus $p : a =_A x$), we have (using the characterisation of $=$ on Σ-types):

$$(a, \mathtt{refl}(a)) =_{\Sigma_{x:A} a =_A x} (x, p)$$

This follows from

$$\mathtt{transport}^{\Pi_{x:A} a =_A x}\, p\, (\mathtt{refl}(a)) =_{a =_A x} (\mathtt{refl}(a)) \cdot p =_{a =_A x} p$$

Alternative (simpler) proof: use based path induction.

Equivalence of propositions

If P and Q are propositions, and P and Q are logically equivalent, then

$$P \overset{\sim}{\longleftrightarrow} Q$$

Indeed, if $f : P \to Q$ and $g : Q \to P$ are given, then $g(fx) =_P x$ holds since P is a proposition (idem Q).

Hence, under the assumption that P and Q are propositions, every pair of functions $f : P \to Q$ and $g : Q \to P$ is a pair of quasi-inverses.

Closure of propositions under some type formers

Type formers preserve propositions. As an example we prove that if $B : A \to \mathcal{U}$ is such that, for all $x : A$, Bx is a proposition, then for any type A we have that $\Pi_{x:A} Bx$ is a proposition.

Indeed, suppose that $f, g : \Pi_{x:A} Bx$. By function extensionality, all we need to prove is that $fx =_{Bx} gx$ for all x, but this holds since Bx is a proposition.

Note that A does not need to be a proposition in the proof above.

Curry-Howard

We can interpret types as propositions, and terms as proofs (but the homotopy type theory perspective says that not all types are propositions).

- One can read $A \times B$ as $A \wedge B$, since in order to prove $A \wedge B$ we need a pair (p, q) where p is a proof of A and q is a proof of B.
- One can read $A \to B$ as $A \Rightarrow B$, since proving $A \Rightarrow B$ amounts to proving B under assumption A: compare with

$$\frac{\Gamma, x : A \vdash p : B}{\Gamma \vdash \lambda x.p : A \to B}$$

- One can read $A + B$ as $A \vee B$, since in order to prove $A \vee B$ we need a proof of A or a proof of B.
- One can read the identity type as the equality predicate.
- One can read Π and Σ as universal and existential quantification, respectively.

Sets

We define a type A to be a set , or 0-type, if $(\mathtt{isSet}\,A)$ is inhabited, where

$$(\mathtt{isSet}\,A) :\equiv \Pi_{x,y:A}\Pi_{p,q:(x=_A y)}(p=q)$$

or equivalently if $\Pi_{x,y:A}(\mathtt{isProp}\,(x =_A y))$.

We then can define a 1-type, or groupoid, to be a type A such that

$$\Pi_{x,y:A}(\mathtt{isSet}\,(x =_A y))$$

Etc. This yields a hierarchy of types, based on how rich their homotopical structure is. We set

$$\begin{aligned}\mathtt{is\text{-}(-1)\text{-}type(A)} &:\equiv \mathtt{isProp(A)}\\ \mathtt{is\text{-}(n+1)\text{-}type(A)} &:\equiv \Pi_{\mathtt{x,y:A}}\mathtt{is\text{-}n\text{-}type(x =_A y)}\end{aligned}$$

If $\mathtt{is\text{-}n\text{-}type(A)}$ is inhabited, we say that A has h-level n or is an n-type. For example, sets and groupoids have h-level 0 and 1.

Characterisations of contractible types

For any type A, the following are logically equivalent:

(1) A is contractible,

(2) A is a proposition and is inhabited, i.e. there exists some $a : A$,

(3) $A \overset{\sim}{\longleftrightarrow} 1$.

We need preliminary facts:

(a) $\square_2$ in the slide "Characterising $=_1$" witnesses that 1 is a proposition.

(b) If A is contractible and $A \overset{\sim}{\longleftrightarrow} B$, then B is contractible. In fact, it holds more generally if B is a retract (notion defined later) of A, as we shall see.

It follows from the statement that under the assumption that A is inhabited (typically when we have $x : A$ in the context), we have that A is a proposition if and only if it is contractible. We shall refer to this as the `isProp` trick.

Characterisations of contractible types (proof)

• (1) $\Rightarrow$ (2) If A is contractible with center a, then it is inhabited by definition, and we have $x =_A a =_A y$ for all $x, y : A$.

• (2) $\Rightarrow$ (1) Let $f : \Pi_{x:A}\Pi_{y:A} x =_A y$. Then $(a, fa) : \mathtt{isContr}(A)$.

• (2) $\Rightarrow$ (3) By (a) and since A and $\mathtt{1}$ are propositions, it is enough to prove that A and $\mathtt{1}$ are logically equivalent. Indeed, we have

$$\lambda x.\star : A \to \mathtt{1} \qquad\qquad \lambda z.a : \mathtt{1} \to A$$

• (3) $\Rightarrow$ (2) We note that $\mathtt{1}$ satisfies (2), and hence (1), so that we can conclude by (b).

Every proposition is a set

Suppose that A is a proposition, i.e. we have f such that $f(x, y) : x =_A y$ for all $x, y : A$.

Our goal is to prove that $x =_A y$ is a proposition. We show that $x =_A y$ is in fact contractible. So we look for a center and a contraction.

For this, we fix $z : A$, and consider the family $C :\equiv \lambda u.(z =_A u)$. We have seen that $(\mathtt{transport}^C pq = q \cdot p)$ for all $p : x =_A y$ and $q : Cx$.

Let $g :\equiv \lambda u.f(z, u)$. We have $g : \Pi_{u:A} Cu$, and hence $(\mathtt{apd}\, g\, p) : g(x) \cdot p =_{z=_A y} g(y)$, from which we deduce

$$p =_{x=_A y} (g(x)^{-1} \cdot g(y))$$

i.e., $g(x)^{-1} \cdot g(y)$ (that depends only on x, y, f) is the center, since this holds for any $p : x =_A y$.

$\mathtt{isContr}(A)$ and $\mathtt{isProp}(A)$ are propositions

• `isProp`: Suppose that $f, g : \Pi_{x,y:A} x =_A y$. By function extensionality, we have to prove that $f(x,y) =_{x=_A y} g(x,y)$ for all $x, y : A$. But under the assumption f (or g), A is a proposition, hence is a set. Therefore $x =_A y$ is a proposition and the sought equality holds for free.

• `isContr`: By induction on Σ-types, it is enough to find for all $a, a' : A$, $\pi : \Pi_{x:A}(a =_A x)$ and $\rho : \Pi_{x:A}(a' =_A x)$:

(i) $q : a =_A a'$, and

(ii) an inhabitant of $(\mathtt{transport}^C\, q\, \pi) =_{Ca'} \rho$ (for $Cz = \Pi_{x:A}(z =_A x)$).

- We define $q = \pi(a')$.
- By function extensionality, we are left to prove

$$\mathtt{transport}^C\, q\, \pi\, z =_{a' =_A z} \rho\, z$$

for all z. But under the assumption π, A is contractible, hence is a proposition, hence is a set, so that this comes for free.

Equivalence of contractible types

• If P and Q are contractible, then they are logically equivalent, and all functions $f : P \to Q$ have quasi-inverses.

Proof: The second part of the statement is an obvious consequence of logical equivalence (as seen before). Let a_0 and b_0 be centers for P and Q: then $\lambda x.b_0$ and $\lambda y.a_0$ witness the two parts of the claimed logical equivalence.

Contractible types and Σ-types

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

(i) If each $P(x)$ is contractible, then $\Sigma_{x:A} P(x)$ is equivalent to A.

Proof: By the assumption, there exists $\mathtt{center} : \Pi_{x:A} P(x)$ such that for all x and $y \in P(x)$ we have $y =_{P(x)} \mathtt{center}(x)$. Then we take

$$\mathtt{pr}_1 : \Sigma_{x:A} P(x) \to A \qquad \lambda x.(x, \mathtt{center}(x)) : A \to \Sigma_{x:A} P(x).$$

(ii) If A is contractible with center a_0, then $\Sigma_{x:A} P(x)$ is equivalent to $P(a_0)$.

Proof: Let (as a first try) $(a_0, f) : \mathtt{isContr}(A)$. We take

$$\begin{aligned} &\lambda b.(a_0, b) : P(a_0) \to \Sigma_{x:A} P(x) \\ &\lambda(a, b).\mathtt{transport}^P\,(fa)^{-1}\,b : \Sigma_{x:A} P(x) \to P(a_0) \end{aligned}$$

We want to check that these maps are quasi-inverses. One direction follows from the characterisation of equality types on Σ-types:

$$(a, b) =_{\Sigma_{x:A} P(x)} (a_0, \mathtt{transport}^P\,(fa)^{-1}\,b) \quad \text{since } fa : a_0 =_A a$$

The other direction is more tricky and requires to make a good choice of f.

Completing the proof of (ii)

(a) First we observe that we can choose f such that $fa_0 =_A \texttt{refl}(a_0)$ (propositionally). Indeed, we can replace an arbitrary f with $\lambda a.(fa_0)^{-1} \cdot fa$:

$$(\lambda a.(fa_0)^{-1} \cdot fa)a_0 \equiv (fa_0)^{-1} \cdot fa_0 =_A \texttt{refl}(a_0)$$

(b) Next, we have the following fact, for any family of types $P : A \to \mathcal{U}$. If $x : A$, if $(p =_{x =_A x} \texttt{refl}(x))$, then for all $y : Px$ we have $(\texttt{transport}^P\, p\, b) = b$. This follows by using based path induction $\texttt{ind}'_{=_{x=_A x}}$.

(c) Finally, we need another easy fact: if $p =_{x =_A x} \texttt{refl}(x)$, then also $p^{-1} =_{x =_A x} \texttt{refl}(x)$. Indeed $p = \texttt{refl}(x)$ implies

$$\texttt{refl}(x) = p^{-1} \cdot p = p^{-1} \cdot \texttt{refl}(x) = p^{-1}$$

We can now complete the proof of (ii): starting from $b : Pa_0$, we get (a_0, b), and then $\texttt{transport}^P\,(fa_0)^{-1}\, b$. But we have chosen in (a) f such that fa_0 (and hence also $(fa_0)^{-1}$ by (c)) is propositionally equal to $\texttt{refl}(a_0)$, and we conclude by (b).

Stability of contractible types under retraction

A retraction is a function $r : A \to B$ such that there exists $s : B \to A$ and a homotopy $\epsilon : r \circ s \sim \mathtt{id}_B$. We say that B is a retract of A (through r, s, ϵ).

The following closure property holds: if B is a retract of A and if A is contractible, then B is contractible.

Proof: Let $(a_0, f) : \mathtt{isContr}(A)$. We take $r(a_0)$ as center. For $y : B$, $f(sy)$ is a path from a_0 to sy, from which we get

$$r(f(sy)) \cdot (\epsilon y) : (r(a_0) =_B y)$$

Hence $(r(a_0), \lambda y.r(f(sy)) \cdot (\epsilon y)) : \mathtt{isContr}(B)$.

$\mathtt{isContr}$ is h-level -2

We have

$$\mathtt{isProp}(A) \overset{\sim}{\longleftrightarrow} \Pi_{x,y:A}\mathtt{isContr}(x =_A y)$$

We first note that both $\mathtt{isProp}(A)$ and $\Pi_{x,y:A}\mathtt{isContr}(x =_A y)$ are propositions (for the latter, we use the fact that $\mathtt{isContr}(x =_A y)$ is a proposition). So we just have to prove a logical equivalence.

- Given $d : \Pi_{x,y:A}\mathtt{isContr}(x =_A y)$, we get $\lambda x, y.\mathtt{pr}_1(d(x, y)) : \mathtt{isProp}(A)$.
- Given $c : \mathtt{isProp}(A)$, and given $x, y : A$, it is enough to provide an inhabitant of $x =_A y$ and an inhabitant of $\mathtt{isProp}(x =_A y)$:
  - for the first, we take $c(x, y)$;
  - for the second, using that A is a set, witnessed by c', we take $c'(x, y)$.

It follows that we can start at h-level -2 and set

$$\mathtt{is}\text{-}(-2)\text{-}\mathtt{type(A)} :\equiv \mathtt{isContr(A)}$$

Cumulativity of h-levels

We shall show that, for all types A, if A is of h-level n, it is also of h-level $n+1$. We prove this universal quantification by induction on n.

• The base case $n = -2$ is clear: a contractible type is a proposition, and we have proved that propositions occupy h-level -1.

• For the inductive case, knowing that A has h-level n, we have to prove $\texttt{is-n-type}(\texttt{x} =_{\texttt{A}} \texttt{y})$ for all x, y: indeed, since by assumption A has h-level n, $x =_A y$ has h-level $n-1$, and hence has h-level n by induction hypothesis.

Closure of n-types under some type formers

n-types are closed under type formers. As an example, we prove hat if $B : A \to \mathcal{U}$ is such that, for all $x : A$, Bx is an n-type, then for any type A we have that $\Pi_{x:A} Bx$ is an n-type.

• Base case $n = -2$. We have seen this for $n = -1$. For $n = -2$, if all Bx are contractible, then a fortiori they are propositions, and hence case $n = -1$ gives us that $\Pi_{x:A} Bx$ is a proposition. It remains to show that $\Pi_{x:A} Bx$ is inhabited. Let $f : \Pi_{x:A}\, \mathtt{isContr}(A)$. Then take $\lambda x.\mathtt{pr}_1(fx)$.

• Inductive case. $f, g : \Pi_{x:A} Bx$. We have to prove that $f =_{\Pi_{x:A} Bx} g$ is an n-type. But function extensionality says that this type is equivalent to $\Pi_{x:A} fx =_{Bx} gx$. By univalence, these types are in fact equal, and hence, we can as well prove that $\Pi_{x:A} fx =_{Bx} gx$ is an n-type, since by transport, for any types C, D and $p : C =_{\mathcal{U}} D$ we have that C is an n-type if and only if D is an n-type. But by induction it is enough to show that $fx =_{Bx} gx$ is an n-type, which follows from our assumption that all Bx are of $(n+1)$-type. One can show that n-types are closed under equivalences without appealing to univalence.

`is-n-type(A)` is a proposition

- We have seen this for h-level -2 (and level -1), so the base case is OK.
- Inductive case. We want to show that $\Pi_{x,y:A}\,\texttt{is-n-type}(\texttt{x} =_{\texttt{A}} \texttt{y})$ is a proposition. By closure of propositions under type formers, it is enough to show that $\texttt{is-n-type}(\texttt{x} =_{\texttt{A}} \texttt{y})$ is a proposition, but this holds by induction.

Four logically equivalent notions of equivalence

We have seen $\mathtt{qinv}(f)$ and $\mathtt{biinv}(f)$, and shown them to be logically equivalent. Here are two more logically equivalent ways to express equivalence of A, B via $f : A \to B$:

- half adjoint equivalence:

$$\mathtt{ishae}(f) :\equiv \Sigma_{g:B\to A, \epsilon:(f\circ g\sim \mathtt{id}), \eta:(g\circ f\sim \mathtt{id})} \Pi_{x:A} \left(f(\eta x) = \epsilon(fx)\right)$$

- contractible map:

$$\mathtt{contrmap}(f) :\equiv \Pi_{y:B}\, \mathtt{isContr}(\mathtt{fib}_f(y))$$

We shall show (remember that $\longleftrightarrow$ is logical equivalence):

$$\begin{aligned} \mathtt{qinv}(f) &\longleftrightarrow \mathtt{ishae}(f) \\ \mathtt{ishae}(f) &\longleftrightarrow \mathtt{contrmap}(f) \end{aligned}$$

It follows that

$$\mathtt{biinv}(f) \longleftrightarrow \mathtt{qinv}(f) \longleftrightarrow \mathtt{ishae}(f) \longleftrightarrow \mathtt{contrmap}(f)$$

Three equivalent notions of equivalence

We shall show that $\mathtt{biinv}(f)$, $\mathtt{ishae}(f)$ and $\mathtt{contrmap}(f)$ are propositions. It will follow that

$$\mathtt{biinv}(f) \overset{\sim}{\longleftrightarrow} \mathtt{ishae}(f) \overset{\sim}{\longleftrightarrow} \mathtt{contrmap}(f)$$

and hence univalence can be formulated replacing `biinv` by `ishae` or `contrmap`.

We shall thus sometimes use a uniform notation $\mathtt{isequiv}(f)$ to stand for any of $\mathtt{biinv}(f)$, $\mathtt{ishae}(f)$ or $\mathtt{contrmap}(f)$, and in fact for any proposition logically equivalent to them.

It can be shown that $\mathtt{qinv}(f)$ is not a proposition, and hence that $\mathtt{qinv}(f)$ is not equivalent to the three other notions.
We shall sketch the proof that using `qinv` in the axiom of univalence would lead to an inconsistent system.

Notions of equivalence in a nutshell

$$\begin{aligned}
&\texttt{biinv}(f) :\equiv (\Sigma_{g:B\to A}(f\circ g \sim \texttt{id})) \times (\Sigma_{h:B\to A}(h\circ f \sim \texttt{id}))\\
&\texttt{qinv}(f) :\equiv \Sigma_{g:B\to A}(f\circ g \sim \texttt{id}) \times (g\circ f \sim \texttt{id})\\
&\texttt{ishae}(f) :\equiv \Sigma_{g:B\to A,\epsilon:(f\circ g\sim\texttt{id}),\eta:(g\circ f\sim\texttt{id})}\Pi_{x:A}\,(f(\eta x) = \epsilon(fx))\\
&\texttt{ishae}'(f) :\equiv \Sigma_{g:B\to A,\epsilon:(f\circ g\sim\texttt{id}),\eta:(g\circ f\sim\texttt{id})}\Pi_{y:B}\,(g(\epsilon x) = \eta(gy))\\
&\texttt{contrmap}(f) :\equiv \Pi_{y:B}\texttt{isContr}(\texttt{fib}_f(y))
\end{aligned}$$

Remarks:

- $\texttt{qinv}(f)$ stands "in the middle" between $\texttt{biinv}(f)$ and $\texttt{ishae}(f)$, in the sense that the logical implications

$$\texttt{ishae}(f) \longrightarrow \texttt{qinv}(f) \longrightarrow \texttt{biinv}(f)$$

  are trivial (forgetful).
- $\texttt{ishae}'$ is symmetric to $\texttt{ishae}$. We call $f(\eta x) = \epsilon(fx)$ or $g(\epsilon x) = \eta(gy)$ coherence equations. It can be shown that requiring the two coherence equations together also gets us into trouble!

$\mathtt{qinv}(f) \to \mathtt{ishae}(f)$

Given $(g, \epsilon, \eta) : \mathtt{qinv}(f)$, we define $(g, \epsilon', \eta, \tau) : \mathtt{ishae}(f)$ as follows:

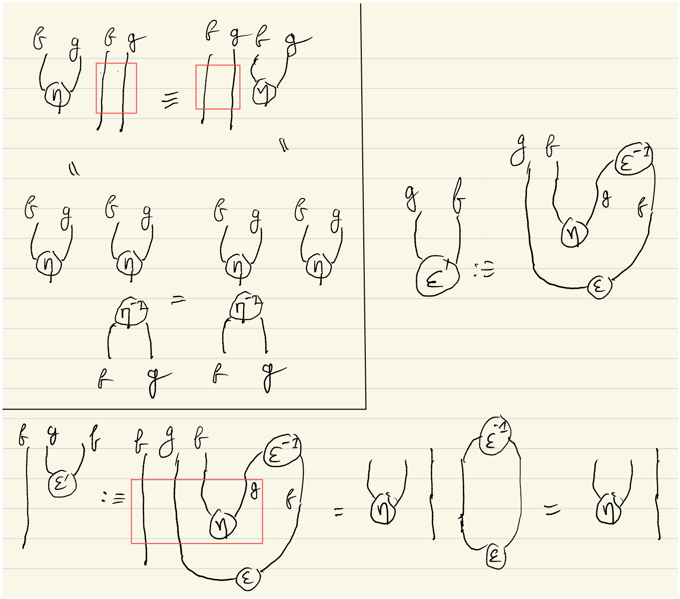
$$\epsilon'(b) = \epsilon(f(g(b)))^{-1} \cdot f(\eta(g(b))) \cdot \epsilon(b)$$

We seek $\tau(a) : \epsilon(f(g(f(a))))^{-1} \cdot f(\eta(g(f(a)))) \cdot \epsilon(f(a)) = f(\eta(a))$.
Indeed, we have

$$\begin{aligned} f(\eta(g(f(a)))) \cdot \epsilon(f(a)) &= f(g(f(\eta(a)))) \cdot \epsilon(f(a)) && \text{(by Exercise 10)} \\ &= \epsilon(f(g(f(a)))) \cdot f(\eta(a)) && \text{(by Exercise 9)} \end{aligned}$$

This is more appealing with string diagrams.

String diagram companion to $\mathtt{qinv}(f) \to \mathtt{ishae}(f)$



$\mathtt{ishae}(f) \longrightarrow \mathtt{contrmap}(f)$

We shall need a characterisation of the equality in $\mathtt{fib}_f(y)$:

$$(\S) \quad (x,p) =_{\mathtt{fib}_f(y)} (x',p') \quad \overset{\sim}{\longleftrightarrow} \quad \Sigma_{\gamma : x =_A x'}(f(\gamma)\cdot p' = p)$$

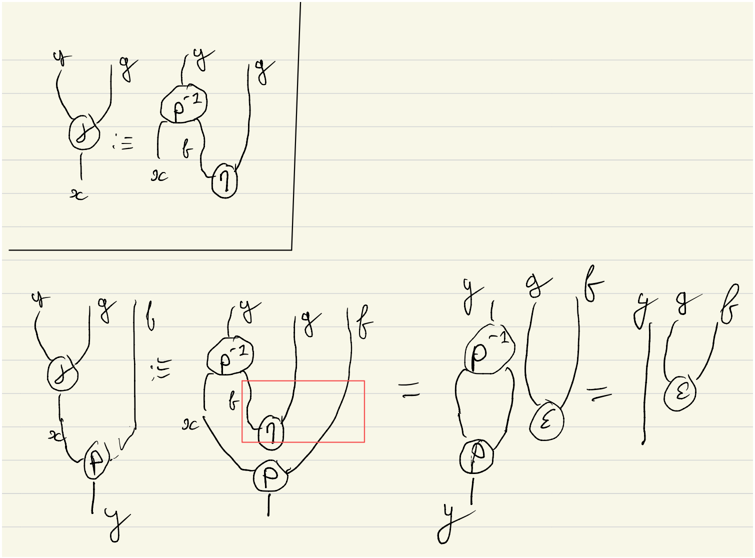
(the proof is a variation on things seen already).

Let $(g,\epsilon,\eta,\tau) : \mathtt{ishae}(f)$ We have to prove that, for all $y : B$, $\mathtt{fib}_f(y)$ is contractible. We choose $(gy,\epsilon y)$ as center of contraction. Let $(x,p) : \mathtt{fib}_f(y)$. We want a path $\gamma : gy =_A x$ such that $f(\gamma)\cdot p = \epsilon y$. We take

$$\gamma :\equiv (g(p))^{-1} \cdot \eta x$$

$$\begin{array}{rcll} f(\gamma)\cdot p & = & (f(g(p)))^{-1} \cdot f(\eta x) \cdot p & \text{(functoriality of } f\text{)} \\ & = & (f(g(p)))^{-1} \cdot \epsilon(fx) \cdot p & \text{(half equivalence)} \\ & = & \epsilon y \cdot p^{-1} \cdot p & \text{(Exercise 9)} \\ & = & \epsilon y & \end{array}$$

String diagram companion to $\mathtt{ishae}(f) \longrightarrow \mathtt{contrmap}(f)$



$\mathtt{ishae}(f) \longleftarrow \mathtt{contrmap}(f)$

Let $P : \mathtt{contrmap}(f)$. We define g, ϵ by

$$gy :\equiv \mathtt{pr}_1(\mathtt{pr}_1(Py)) \qquad \epsilon y :\equiv \mathtt{pr}_2(\mathtt{pr}_1(Py))$$

We are left to exhibit an inhabitant of $\Sigma_{\eta : g\circ f \sim \mathtt{id}} \Pi_{x:A}\, f(\eta x) = \epsilon(fx)$. We can repackage the information needed as follows:

$$(\P) \quad \begin{aligned} &\Sigma_{\eta : g\circ f \sim \mathtt{id}} \Pi_{x:A}\, f(\eta x) = \epsilon(fx) \\ &\overset{\sim}{\longleftrightarrow}\ \Pi_{x:A}\, (g(fx), \epsilon(fx)) =_{\mathtt{fib}_f(fx)} (x, \mathtt{refl}(fx)) \end{aligned}$$

Proof of (¶): By (§), we retrieve ηx and $f(\eta x) = f(\eta x) \cdot \mathtt{refl}(fx) = \epsilon(fx)$ from $(g(fx), \epsilon(fx)) =_{\mathtt{fib}_f(fx)} (x, \mathtt{refl}(fx))$.

Back to the proof of $\mathtt{ishae}(f)$: By (¶), we are left to provide a path from $(x, \mathtt{refl}(fx))$ to $(g(fx), \epsilon(fx))$ in $\mathtt{fib}_f(fx)$. But this is a consequence of our assumption that $\mathtt{fib}_f(fx)$ is contractible. Explicitly, we take

$$(\mathtt{pr}_2(P(fx))(x, \mathtt{refl}(x)))^{-1} \cdot (\mathtt{pr}_2(P(fx))(g(fx), \epsilon(fx)))$$

isProp(contrmap(f))

That $\mathtt{contrmap}(f)$ is a proposition follows obviously from $\mathtt{isContr}(A) \longrightarrow \mathtt{isProp}(A)$ (applied to all fibers of f) and from the closure of propositions under type formers.

For the other two assertions isProp(biinv(f)) and isProp(ishae(f)), we shall use the following property:

$$(\dagger\dagger) \quad \mathtt{qinv}(f) \longrightarrow \mathtt{isContr}(\Sigma_{g:B\to A}(g \circ f \sim \mathtt{id}_A))$$

Proof: We have that $\Sigma_{g:B\to A}(g \circ f \sim \mathtt{id}_A)$ is $\mathtt{fib}_{-\circ f}(\mathtt{id}_A)$. We next observe that if g is a quasi-inverse of f, then $(- \circ g)$ is a quasi-inverse of $(- \circ f)$. Hence, by the logical implications proved, $(- \circ f)$ is a contractible map, which proves a fortiori the above implication.

isProp(biinv(f)) and isProp(ishae(f))

- $\mathtt{isProp}(\mathtt{biinv}(\mathrm{f}))$: We use the $\mathtt{isProp}$ trick, i.e., we assume $\mathtt{biinv}(f)$ and prove that $\mathtt{biinv}(f)$ is contractible. But since $\mathtt{biinv}(f) \longrightarrow \mathtt{qinv}(f)$, we can apply (††) and get $\mathtt{isContr}(\Sigma_{g:B\to A}(g \circ f \sim \mathtt{id}_A))$. One gets similarly $\mathtt{isContr}(\Sigma_{h:B\to A}(f \circ h \sim \mathtt{id}_A))$, and we conclude by closure of contractible types under product.

- $\mathtt{isProp}(\mathtt{ishae}(\mathrm{f}))$: We may again assume $\mathtt{ishae}(f)$. By (¶), we get

$$\mathtt{ishae}(f) \overset{\sim}{\longleftrightarrow} \Sigma_{u:S} Tu \quad \text{where}$$

$S :\equiv \Sigma_{h:B\to A}(f \circ h \sim \mathtt{id}_A)$
$Tu :\equiv \Pi_{x:A}\,(g(fx), \epsilon(fx)) =_{\mathtt{fib}_f(fx)} (x, \mathtt{refl}(fx))$ (for $g \equiv \mathtt{pr}_1(u), \epsilon \equiv \mathtt{pr}_2(u)$)

We have that

- S is contractible by (the symmetric version of) of (††) (since $\mathtt{biinv}(f) \longrightarrow \mathtt{qinv}(f)$);
- Tu is contractible (for all u): we first note that $\mathtt{fib}_f(fx)$ is contractible since $\mathtt{biinv}(f) \longrightarrow \mathtt{contrmap}(f)$). Then we apply Exercise 12 and the closure property under Π-types of contractibility.

We conclude by Exercise 13.

The black sheep qinv

• One can show (Section 4.1 of the HoTT book) that $\mathtt{qinv}(f)$ is not always a proposition.

• Let us attempt to reformulate the univalence axiom using qinv instead of ishae (or biinv, contrmap), i.e., let us assume ("black sheep univalence") the existence of a quasi-inverse $\mathtt{ua}'$ for the canonical map $\mathtt{idtoeqv}' : (A =_{\mathcal{U}} B) \to \Sigma_{f:A\to B}\, \mathtt{qinv}(f)$. One can show that this version of univalence still works to establish that $\mathtt{qinv}(f)$ is not always a proposition.

• We shall sketch the proof that this univalence axiom is inconsistent. The proof relies on the fact the property that $\mathtt{ishae}(f)$ is a proposition, established earlier (without making use of a univalence axiom).

Inconsistency of univalence based on the black sheep

By definition of $\mathtt{idtoeqv}'$ and $\mathtt{idtoeqv}$ (adapted to $\Sigma_{f:A\to B}\mathtt{ishae}(f)$), we have:

$$(\circ)\quad \mathtt{idtoeqv}'(\pi) = \mathtt{pr}(\mathtt{idtoeqv}(\pi)) \quad \text{for all } \pi : A =_{\mathcal{U}} B$$

where $\mathtt{pr} : \Sigma_{f:A\to B}\mathtt{ishae}(f) \to \Sigma_{f:A\to B}\mathtt{qinv}(f)$ is the projection map. We shall show $\mathtt{isProp}(\mathtt{qinv}(f))$ for all $f : A \to B$, contradicting previous slide. Let

$$(g_i, \eta_i, \epsilon_i) : \mathtt{qinv}(f) \text{ and } (f_i', g_i', \eta_i', \epsilon_i', \tau_i) :\equiv \mathtt{idtoeqv}(\mathtt{ua}'(f, g_i, \eta_i, \epsilon_i)) \quad (i = 1, 2)$$

Then by $(\circ)$ and "black sheep univalence" we get

$$(f, g_i, \eta_i, \epsilon_i) = (f_i', g_i', \eta_i', \epsilon_i') \quad (i = 1, 2)$$

In particular, we have $R : f = f_1'$ and $S : f = f_2'$, and similarly $g_1 = g_1'$ and $g_2 = g_2'$. Since $R^{-1} \cdot S : f_1' = f_2'$, we can use transport to get $g', \eta', \epsilon', \tau$ such that

$$(f_1', g', \eta', \epsilon', \tau) = (f_2', g_2', \eta_2', \epsilon_2', \tau_2)$$

Since $(g_1', \eta_1', \epsilon_1', \tau_1), (g', \eta', \epsilon', \tau) : \mathtt{ishae}(f_1')$ and since $\mathtt{ishae}(f_1')$ is a proposition, we get

$$(g_1', \eta_1', \epsilon_1', \tau_1) = (g', \eta', \epsilon', \tau) \quad \text{and a fortiori } (g_1', \eta_1', \epsilon_1') = (g', \eta', \epsilon')$$

Collecting this all, we constitute a chain that will allow us to conclude (next slide):

$$(f, g_1, \eta_1, \epsilon_1) = (f_1', g_1', \eta_1', \epsilon_1') = (f_1', g', \eta', \epsilon') = (f_2', g_2', \eta_2', \epsilon_2') = (f, g_2, \eta_2, \epsilon_2)$$

Rounding up the proof of $\mathtt{isProp}(\mathtt{qinv}(f))$ in presence of the black sheep

We monitor the chain

$$(f, g_1, \eta_1, \epsilon_1) \overset{\text{black sheep}}{=} (f_1', g_1', \eta_1', \epsilon_1') \overset{\mathtt{isProp}(\mathtt{ishae}(f_1'))}{=} (f_1', g', \eta', \epsilon') \overset{\text{transport}}{=} (f_2', g_2', \eta_2', \epsilon_2') \overset{\text{black sheep}}{=} (f, g_2, \eta_2, \epsilon_2)$$

more closely, through the following table (omitting the ϵ's):

$$\begin{array}{ccccccccc} f & \overset{R}{=} & f_1' & \overset{\mathtt{refl}}{=} & f_1' & \overset{R^{-1}\cdot S}{=} & f_2' & \overset{S^{-1}}{=} & f \\ g_1 & = & g_1' & = & g' & = & g_2' & = & g_2 \\ \eta_1 & & \eta_1' & & \eta' & & \eta_2' & & \eta_2 \end{array}$$

We then have, for all x

$$\begin{array}{rcll} (\eta_1 x) & = & p_1 \cdot (\eta_1' x) & (p_1 : g_1(fx) = g_1'(f_1' x)) \\ (\eta_1' x) & = & p_2 \cdot (\eta' x) & (p_2 : g_1'(f_1' x) = g'(f_1' x)) \\ (\eta' x) & = & p_3 \cdot (\eta_2' x) & (p_3 : g'(f_1' x) = g_2'(f_2' x)) \\ (\eta_2' x) & = & p_4 \cdot (\eta_2 x) & (p_4 : g_2'(f_2' x) = g_2(fx)) \end{array}$$

and therefore $(\eta_1 x) = p' \cdot (\eta_2 x)$, where $p' :\equiv p_1 \cdot p_2 \cdot p_3 \cdot p_4$. Our goal is to prove $(\eta_1 x) = p \cdot (\eta_2 x)$, where $p : g_1(fx) = g_2(fx)$ is induced by $g_1 = g_2$ (through g_1', g', g_2'). In contrast, p' is induced by the same proof of $g_1 = g_2$ and by (cf. table above) $T :\equiv R \cdot \mathtt{refl} \cdot (R^{-1} \cdot S) \cdot S^{-1}$. Since $T = \mathtt{refl}$, we deduce easily $p = p'$, which in turn implies the goal.

Fiberwise maps

Let $P, Q : A \to \mathcal{U}$ and $f : \Pi_{x:A} Px \to Qx$ (fiberwise map from P to Q). Then f induces a map $\mathtt{total}(f) : \Sigma_{x:A} Px \to \Sigma_{x:A} Qx$ defined by

$$\mathtt{total}(f)(a, u) = (a, f\ a\ u)$$

We have, for all (fixed) $x : A$ and $v : Qx$:

$$\mathtt{fib}_{\mathtt{total}(f)}(x, v) \overset{\sim}{\longleftrightarrow} \mathtt{fib}_{f(x)}(v)$$

Proof: We have

$$\begin{aligned}
&\mathtt{fib}_{\mathtt{total}(f)}(x, v) \\
&\quad \overset{\sim}{\longleftrightarrow} \quad \Sigma_{a:A}\Sigma_{u:Pa}(a, f\ a\ u) = (x, v) \\
&\quad \overset{\sim}{\longleftrightarrow} \quad \Sigma_{a:A}\Sigma_{u:Pa}\Sigma_{p:a=x}\, \mathtt{transport}\, p\, (f\ a\ u) = v \quad \text{(equality in } \Sigma\text{-types)} \\
&\quad \overset{\sim}{\longleftrightarrow} \quad \Sigma_{a:A}\Sigma_{p:a=x}\Sigma_{u:Pa}\, \mathtt{transport}\, p\, (f\ a\ u) = v \\
&\quad \overset{\sim}{\longleftrightarrow} \quad \Sigma_{u:Px}\mathtt{transport}\, \mathtt{refl}(x)\, (f\ x\ u) = v \quad (\mathtt{isContr}(\Sigma_{a:A} a = x)) \\
&\quad \equiv \quad \Sigma_{u:Px}\, f\ x\ u = v \\
&\quad \equiv \quad \mathtt{fib}_{f(x)}(v)
\end{aligned}$$

It follows that $\mathtt{total}(f)$ is a weak equivalence if and only each fx is an equivalence.

Univalence and η-rule implies function extensionality

We want to show that $\mathtt{happly}(f,g) : (f = g) \to (f \sim g)$ (defined before) is an equivalence, for all $f, g : \Pi_{x:A} Px$. We fix f. We set

$$Qg :\equiv (f = g) \qquad Rg :\equiv (f \sim g)$$

$\mathtt{happly}(f,-)$ is a fiberwise map from Q to R. By what we have just seen, our goal reformulates as proving that the total map induced, of type

$$\Sigma_{g:\Pi_{x:A}Px}(f = g) \to \Sigma_{g:\Pi_{x:A}Px}(f \sim g) \equiv \Sigma_{g:\Pi_{x:A}Px}\Pi_{x:A}(fx = gx)$$

is an equivalence. By Exercise 11, $\Sigma_{g:\Pi_{x:A}Px}(f = g)$ is contractible. By our analysis of equivalences between contractible types, it is enough to show that $\Sigma_{g:\Pi_{x:A}Px}(f \sim g)$ is contractible. We shall prove this in two steps:

(i) $\Sigma_{g:\Pi_{x:A}Px}(f \sim g)$ is a retract of $\Pi_{x:A}\Sigma_{u:Px}(fx = u)$

(ii) The property that a product of contractible types is contractible can be established using univalence and not using function extensionality.

Then we can conclude: by Exercise 11 again, $\Sigma_{u:Px}(fx = u)$ is contractible, and a retract of a contractible type is contractible.

$\Sigma_{g:\Pi_{x:A}Px}(f \sim g)$ is a retract of $\Pi_{x:A}\Sigma_{u:Px}(fx = u)$

More generally, let $A : \mathcal{U}$ $P : A \to \mathcal{U}$ and $Q : \Pi_{x:A}(Px \to \mathcal{U})$. Then

$$(\|\|\|) \quad \Sigma_{g:\Pi_{x:A}Px}\Pi_{x:A}\, Q\, x\, (gx) \quad \text{is a retract of} \quad \Pi_{x:A}\Sigma_{u:Px}\, Q\, x\, u$$

We define maps in the two directions as follows:

$$\phi(g,h)x :\equiv (gx, hx) \qquad \psi(k) = (\lambda x.\mathtt{pr}_1(kx), \lambda x.\mathtt{pr}_2(kx))$$

(the type of $\mathtt{pr}_2(kx)$ is $Q\, x\, (\mathtt{pr}_1\, (kx)) \equiv Q\, x\, (\lambda x.(\mathtt{pr}_1\, (kx)))\, x)$).
We have $\psi(\phi(g,h)) :\equiv (\lambda x.gx, \lambda x.hx) = (g,h)$.

Our claim (1) then follows by instantiation of $(\|\|\|)$ with $Qxy :\equiv (fx = y)$.

Note the use of the (here assumed) propositional η-rule, namely

$$\lambda x.fx = f \qquad \text{(for } x \text{ not free in } f\text{)}$$

Remarks:

- Propositional η follows from function extensionality.
- We have $\phi(\psi(f)) \sim f$ for all f, and hence, if function extensionality is assumed, or proved, we have $\phi \circ \psi \sim \mathtt{id}$, and hence $\Sigma_{g:\Pi_{x:A}Px}(f \sim g)$ is in fact in the end equivalent to $\Pi_{x:A}\Sigma_{u:Px}(fx = u)$.

Plan for the proof of (ii)

Let $P : A \to \mathcal{U}$ be a family of contractible types. We have seen earlier that $\mathtt{pr}_1 : (\Sigma_{x:A} Px) \to A$ is an equivalence (and this does not use function extensionality).

We shall prove the following properties

(a) $\Pi_{x:A} Px$ is a retract of $\mathtt{fib}_\alpha(\mathtt{id}_A)$, where

$$\alpha :\equiv \lambda fx.\mathtt{pr}_1(fx) : (A \to \Sigma_{x:A} Px) \to (A \to A)$$

(b) α is an equivalence.

Then we can conclude, because α being an equivalence can be expressed as $\mathtt{contrmap}(\alpha)$, hence $\Pi_{x:A} Px$ is a retract of a contractible type and therefore is contractible.

Proof of (a)

(a) $\Pi_{x:A} Px$ is a retract of $\mathtt{fib}_\alpha(\mathtt{id}_A)$, where

$$\alpha :\equiv \lambda f x.\mathtt{pr}_1(fx) : (A \to \Sigma_{x:A} Px) \to (A \to A)$$

We define maps in the two directions:

$$\phi(f) :\equiv (\lambda x.(x, fx), \mathtt{refl}(\mathtt{id}_A)) : \Pi_{x:A} Px \to \mathtt{fib}_\alpha(\mathtt{id}_A)$$
$$\psi(g,p) :\equiv \lambda x.\mathtt{transport}\,(\mathtt{happly}\, p\, x)\,(\mathtt{pr}_2(gx)) : \mathtt{fib}_\alpha(\mathtt{id}_A) \to \Pi_{x:A} Px$$

The definition of ψ type-checks because

$$\alpha(g)x \equiv \mathtt{pr}_1(gx) \qquad \text{and} \qquad \mathtt{pr}_2(gx) : P(\mathtt{pr}_1(gx))$$

We have:

$$\begin{aligned} \psi(\phi(f)) \;&\equiv\; \psi(\lambda x.(x, fx), \mathtt{refl}(\mathtt{id}_A)) \;\equiv\; \lambda x.\mathtt{transport}\,(\mathtt{refl}(x))(fx) \\ &\phantom{\equiv\; \psi(\lambda x.(x, fx), \mathtt{refl}(\mathtt{id}_A))}\;\equiv\; \lambda x.fx \quad = \quad f \end{aligned}$$

(since $\mathtt{happly}\, p\,(\mathtt{refl}(\mathtt{id}_A)) \equiv \mathtt{refl}(x)$ and $\mathtt{pr}_2((\lambda x.(x, fx))x) \equiv fx$).

Note the use of the propositional η-rule again.

Plan for the proof of (b)

(b) α is an equivalence, where

$$\alpha :\equiv \lambda fx.\mathtt{pr}_1(fx) : (A \to \Sigma_{x:A} Px) \to (A \to A)$$

This is where we make use of univalence!

More generally, we prove that if $e : A \to B$ is an equivalence, then so is $\lambda f\, x.\, e(fx) : (X \to A) \to (X \to B)$. This goes in two steps:

(b_1) We show this in the case $e :\equiv \mathtt{idtoeqv}(p)$ (for $p : A = B$).

(b_2) We show that if $e_1 = e_2$ and $\mathtt{isequiv}(\lambda f\, x.\, e_1(fx))$, then $\mathtt{isequiv}(\lambda f\, x.\, e_2(fx))$.

We then conclude since, by univalence, we have $\mathtt{idtoeqv}(\mathtt{ua}(e)) = e$. So, setting $e_1 \equiv \mathtt{idtoeqv}(\mathtt{ua}(e))$ and $e_2 :\equiv e$, we can apply (b_1) and then (b_2).

Completing the proof of (b)

(b_1) $\lambda f\,x.\,\texttt{idtoeqv(p)}\,(fx) : (X \to A) \to (X \to B)$ is an equivalence.

• (b_1) is proved by path induction. We are reduced to prove $\texttt{isequiv}(\lambda f\,x.\,(\texttt{idtoeqv}(\texttt{refl}))(fx))$, which is trivial since

$$\lambda f\,x.\,(\texttt{idtoeqv}(\texttt{refl}))(fx) \equiv \lambda f\,x.\,fx = \lambda f.f$$

(identity functions have quasi-inverses!). We use again the η-rule!

(b_2) If $e_1 = e_2$ and $\texttt{isequiv}(\lambda f\,x.\,e_1(fx))$, then $\texttt{isequiv}(\lambda f\,x.\,e_2(fx))$.

• (b_2) is proved by transport. Consider the family $P : (A \to B) \to \mathcal{U}$ defined by $Pe :\equiv \texttt{isequiv}(\lambda f\,x.\,e(fx))$, and let $p : e_1 = e_2$. Then $\texttt{transport}^P p$ turns proofs of $\texttt{isequiv}(\lambda f\,x.\,e_1(fx))$ into proofs of $\texttt{isequiv}(\lambda f\,x.\,e_2(fx))$

This finally completes the proof that (univalence $+\ \eta$) implies function extensionality.

Some properties of $=_{A+B}$

One can prove the following equivalences of types:

$$((\mathtt{inl}\, a_1) =_{A+B} (\mathtt{inl}\, a_2)) \overset{\sim}{\longleftrightarrow} (a_1 =_A a_2)$$
$$((\mathtt{inr}\, b_1) =_{A+B} (\mathtt{inr}\, b_2)) \overset{\sim}{\longleftrightarrow} (b_1 =_A b_2)$$
$$((\mathtt{inl}\, a_1) =_{A+B} (\mathtt{inr}\, b_1)) \overset{\sim}{\longleftrightarrow} 0$$

by the following method, called "encode-decode". We define a family $\mathtt{code} : A + B \to \mathcal{U}$ (tailored to the problem we want to solve) by induction on coproduct as follows ($a_1 : A$ is fixed):

$$\mathtt{code}(\mathtt{inl}\, a) :\equiv (a_1 =_A a) \qquad \mathtt{code}\,(\mathtt{inr}\, b) :\equiv 0$$

Then we show, for all $z : A + B$ that

$$(\mathtt{code}\, z) \overset{\sim}{\longleftrightarrow} ((\mathtt{inl}\, a_1) =_{A+B} z)$$

The first and the third properties listed then follow from instantiation with $z \equiv (\mathtt{inl}\, a_2)$ and $(\mathtt{inr}\, b_1)$, respectively.

The encode-decode method

$$\mathtt{code} : A+B \to \mathcal{U} \qquad \mathtt{code}(\mathtt{inl}\, a) :\equiv (a_1 =_A a) \qquad \mathtt{code}\,(\mathtt{inr}\, b) :\equiv 0$$

We exhibit:

- $\mathtt{encode} : \Pi_{z:A+B}\,((\mathtt{inl}\, a_1) = z) \to (\mathtt{code}\, z)$

$$\mathtt{encode}\, z\, p :\equiv \mathtt{transport}^{\mathtt{code}}\, p\,(\mathtt{refl}(a_1))$$

- $\mathtt{decode} : \Pi_{z:A+B}\,(\mathtt{code}\, z) \to ((\mathtt{inl}\, a_1) = z)$ is defined by induction on the sum type:

$$\begin{aligned}\mathtt{decode}\,(\mathtt{inl}\, a) &:\equiv \mathtt{ap}\,\mathtt{inl}\\ \mathtt{decode}\,(\mathtt{inr}\, b) &:\equiv \mathtt{ind}_0\, C\end{aligned}$$

where in the last clause $C : 0 \to \mathcal{U}$ is the constant $(\mathtt{inl}\, a_1) = (\mathtt{inr}\, b)$.

$(\mathtt{encode}\, z)$ and $(\mathtt{decode}\, z)$ witness an equivalence

$$\mathtt{decode} : \Pi_{z:A+B}\, (\mathtt{code}\, z) \to ((\mathtt{inl}\, a_1) = z)$$
$$\mathtt{encode}\, z\, p :\equiv \mathtt{transport}^{\mathtt{code}}\, p\, (\mathtt{refl}(a_1)) \qquad (p : (\mathtt{inl}(a_1) = z))$$
$$\mathtt{decode}\, (\mathtt{inl}\, a) :\equiv \mathtt{ap}\, \mathtt{inl} \qquad \mathtt{decode}\, (\mathtt{inr}\, b) :\equiv \mathtt{ind}_0\, C$$

• By using $\mathtt{ind}'_{=}$, $\mathtt{decode}\, z\, (\mathtt{transport}^{\mathtt{code}}\, p\, (\mathtt{refl}(a_1))) = p$ reduces to the case where $z \equiv (\mathtt{inl}\, a_1)$ and $p \equiv (\mathtt{refl}(\mathtt{inl}\, a_1))$, in which case we indeed have

$$\begin{aligned}&\mathtt{decode}\, (\mathtt{inl}\, a_1)\, (\mathtt{transport}^{\mathtt{code}}\, (\mathtt{refl}\, (\mathtt{inl}\, a_1))(\mathtt{refl}(a_1)))\\ &\quad \equiv \mathtt{decode}\, (\mathtt{inl}\, a_1)\, (\mathtt{refl}(a_1)) \equiv \mathtt{ap}\, \mathtt{inl}\, (\mathtt{refl}(a_1)) \equiv (\mathtt{refl}(\mathtt{inl}\, a_1))\end{aligned}$$

• To prove $\mathtt{encode}\, z\, (\mathtt{decode}\, z\, u) = u$, we proceed by induction on z. If $z \equiv (\mathtt{inr}\, b)$, then we are done by induction on $\mathtt{0}$. If $z \equiv (\mathtt{inl}\, a)$, then we have $u : (a_1 =_A a)$, and:

$$\begin{aligned}&\mathtt{encode}\, z\, (\mathtt{decode}\, z\, u) \equiv \mathtt{transport}^{\mathtt{code}}\, (\mathtt{ap}\, \mathtt{inl}\, u)\, (\mathtt{refl}(a_1))\\ &= \ \mathtt{transport}^{\mathtt{code}\,\circ\, \mathtt{inl}}\, u\, (\mathtt{refl}(a_1)) = (\mathtt{refl}(a_1)) \cdot u = u\\ &(\text{since } \mathtt{code}\, \circ\, \mathtt{inl} \equiv \lambda z.(a_1 = z))\end{aligned}$$

The circle

- A type S^1
- There are two constructors:

$$\begin{aligned} &\texttt{base} : S^1 \\ &\texttt{loop} : (\texttt{base} =_{S^1} \texttt{base}) \end{aligned}$$

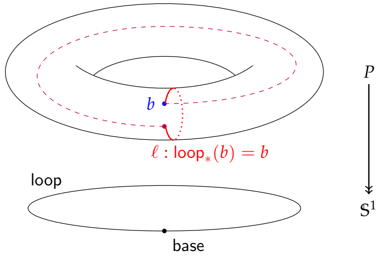
Induction principle:

$$\texttt{ind}_{S^1} : \\ \quad \Pi_{P:S^1 \to \mathcal{U}} \left(\Sigma_{b:P(\texttt{base})} ((\texttt{transport}^P \, \texttt{loop} \, b) =_{P(\texttt{base})} b)\right) \to (\Pi_{x:S^1} Px)$$

Associated equalities:

$$\begin{aligned} &\texttt{ind}_{S^1} P \, (b, l) \, \texttt{base} \equiv b && \text{(definitional)} \\ &\texttt{apd} \, (\texttt{ind}_{S^1} \, P \, (b, l)) \, \texttt{loop} = l && \text{(propositional)} \end{aligned}$$

Illustrating and explaining the induction principle for S^1



In the picture, P on the right should be read as $\Sigma_{x:S^1} Px$. The vertical arrow into S^1 is $\mathtt{pr}_1$, and the torus on the left stands for $\Sigma_{x:S^1} Px$.
We have seen that every path living in the space at the bottom induces a path in the total space. We instantiate this with $\mathtt{loop} : \mathsf{base} = \mathsf{base}$ and $b : P(\mathsf{base})$, and we get the path (represented with dashes)

$$(\mathtt{loop}, \mathtt{refl}(\mathtt{transport}^P\, \mathtt{loop}\, b)) : (\mathsf{base}, b) =_{\Sigma_{x:X} Px} (\mathsf{base}, (\mathtt{transport}^P\, \mathtt{loop}\, b))$$

This path has no reason to be a loop, but a $l : ((\mathtt{transport}^P\, \mathtt{loop}\, b) =_{P(\mathsf{base})} b)$ will complete it (dotted path closing the circle):

$$(\mathtt{refl}(\mathsf{base}), l) : (\mathsf{base}, (\mathtt{transport}^P\, \mathtt{loop}\, b)))) = (\mathsf{base}, b)$$

The recursion principle for S^1

We shall also need a recursion principle for S^1 for a constant family $C :\equiv \lambda x.A$, for some $A : \mathcal{U}$, i.e. we shall use

$$\texttt{rec}_{S^1} : \Pi_{A:\mathcal{U}}\,(\Sigma_{b:A}\,(b =_A b)) \to (S^1 \to A)$$

Associated equalities:

$$\begin{array}{ll} \texttt{rec}_{S^1}\,A\,(b,l)\,\texttt{base} \equiv b & \text{(definitional)} \\ \texttt{ap}\,(\texttt{rec}_{S^1}\,A\,(b,l))\,\texttt{loop} = l & \text{(propositional)} \end{array}$$

That $\texttt{rec}_{S^1}$ can be encoded from $\texttt{ind}_{S^1}$ is proved via an adjustment similar to the adjustment from `apd` to `ap` discussed before.

Here, $\mathcal{U}$ stands for any universe. We shall soon instantiate it to $\mathcal{U}_1$.

The type $\mathbb{Z}$ of integers

We define first the type $\mathbb{N}^+$ of strictly positive natural numbers exactly as `Nat`, replacing the constructor 0 by a constructor 1 (of course it is equivalent to $\mathbb{N}$, but we reserve the symbol 0 for the next step!).

We now define the integers:

- A type $\mathbb{Z}$
- The constructors are

$$\begin{aligned} &0 : \mathbb{Z} \\ &+ : \mathbb{N}^+ \to \mathbb{Z} \\ &- : \mathbb{N}^+ \to \mathbb{Z} \end{aligned}$$

Induction principe:

$$\mathtt{ind}_{\mathbb{Z}} : \Pi_{C:\mathbb{Z}\to\mathcal{U}}\, C_0 \to (\Pi_{x:\mathbb{N}^+} C(+x)) \to (\Pi_{x:\mathbb{N}^+} C(-x)) \to (\Pi_{z:\mathbb{Z}}\, Cz)$$

This principle allows us to define two functions that we shall need (Exercise 15):

$$\begin{aligned} &\lambda z.z + 1 : \mathbb{Z} \to \mathbb{Z} \\ &\lambda z.\,\mathtt{loop}^z : \mathbb{Z} \to (\mathtt{base} =_{S^1} \mathtt{base}) \end{aligned}$$

$\lambda z.z + 1$ and $\lambda z.\,\mathtt{loop}^z$, informally

- We have

$$\begin{aligned}
&0 + 1 \equiv 1\\
&(+1) + 1 \equiv +(\mathtt{succ}(1))\\
&(+(\mathtt{succ}(n))) + 1 \equiv = +(\mathtt{succ}(\mathtt{succ}(n)))\\
&(-1) + 1 \equiv 0\\
&(-(\mathtt{succ}(n))) + 1 \equiv -n
\end{aligned}$$

- We have

$$\begin{aligned}
&\mathtt{loop}^0 \equiv \mathtt{refl}_{\mathtt{base}}\\
&\mathtt{loop}^{+1} \equiv \mathtt{loop}\\
&\mathtt{loop}^{+(\mathtt{succ}(n))} \equiv \mathtt{loop} \cdot \mathtt{loop}^{+n}\\
&\mathtt{loop}^{-1} \equiv \mathtt{loop}^{-1} && \text{(whence the notation } \mathtt{loop}^z\text{)}\\
&\mathtt{loop}^{-(\mathtt{succ}(n))} \equiv \mathtt{loop}^{-1} \cdot \mathtt{loop}^{-n}
\end{aligned}$$

The fundamental group of the circle

Our goal is to prove

$$(\texttt{base} =_{S^1} \texttt{base}) \overset{\sim}{\longleftrightarrow} \mathbb{Z}$$

Here, $(\texttt{base} =_{S^1} \texttt{base})$ (or rather its truncation, see final slides) is the HoTT formulation of the fundamental group $\pi_1(S^1)$: its inhabitants are the loops on base.

We shall use the encode-decode method, and this starts by generalising the statement. We define $\texttt{Eq}_{S^1} : S^1 \to \mathcal{U}$ by instantiating

$$\texttt{rec}_{S^1} : \Pi_{A:\mathcal{U}_1}\,(\Sigma_{b:A}\,(b =_A b)) \to (S^1 \to A)$$

with $A = \mathcal{U}_0$, $b = \mathbb{Z}$, and $\texttt{ua}(\lambda z.z+1) : \mathbb{Z} =_{\mathcal{U}} \mathbb{Z}$, i.e. we set

$$\texttt{Eq}_{S^1} :\equiv \texttt{rec}_{S^1}\,\mathcal{U}_0\,(\mathbb{Z}, \texttt{ua}(\lambda z.z+1)),$$

and hence we have

$$\texttt{Eq}_{S^1}(\texttt{base}) \equiv \mathbb{Z} \qquad (\texttt{ap}\,\texttt{Eq}_{S^1}\,\texttt{loop}) = \texttt{ua}(\lambda z.z+1)$$

We shall prove, for all $x : S^1$

$$(\texttt{base} =_{S^1} x) \overset{\sim}{\longleftrightarrow} \texttt{Eq}_{S^1}(x)$$

Yet another property of transport

For $B : A \to \mathcal{U}$ and $p : x =_A y$, we have the following property (that does not require univalence):

$$\mathtt{transport}^B p =_{Bx \to By} \mathtt{idtoeqv}(\mathtt{ap}\, B\, p)$$

This is proved by path induction: indeed, taking $\mathtt{refl}(x)$ for p, we have $\mathtt{transport}^B p \equiv \mathtt{id}_{Bx}$, $\mathtt{ap} B \mathtt{refl}(x) \equiv \mathtt{refl}(Bx)$ and $\mathtt{idtoeqv}(\mathtt{refl}(Bx)) \equiv \mathtt{id}_{Bx}$.

We apply this general result to $\mathtt{Eq}_{S^1}$ and $\mathtt{loop}$ and obtain

$$\begin{array}{rcl} \mathtt{transport}^{\mathtt{Eq}_{S^1}}\, \mathtt{loop} & = & \mathtt{idtoeqv}(\mathtt{ap}\, \mathtt{Eq}_{S^1}\, \mathtt{loop}) \\ & = & \mathtt{idtoeqv}(\mathtt{ua}(\lambda z.z+1)) \\ & = & \lambda z.z+1 \end{array}$$

And hence, via $\mathtt{happly}$, we get, for all $z : \mathbb{Z}$:

$$(*) \quad \mathtt{transport}^{\mathtt{Eq}_{S^1}}\, \mathtt{loop}\, z = z+1$$

The encode and decode functions

• For $x : S^1$, define $\mathtt{encode}(x) : (\mathtt{base} = x) \to \mathtt{Eq}_{S^1}(x)$ by path induction:

$$\mathtt{encode}(\mathtt{base})(\mathtt{refl}(\mathtt{base})) :\equiv 0 : \mathtt{Eq}_{S^1}(\mathtt{base})$$

• We synthesise a definition of $\mathtt{decode} : \Pi_{x:S^1} (\mathtt{Eq}_{S^1}(x) \to (\mathtt{base} = x))$ by induction on S^1:

$$\mathtt{decode} :\equiv \mathtt{ind}_{S^1}\, P\, \Box$$

where $P :\equiv \lambda x.(\mathtt{Eq}_{S^1}(x) \to (\mathtt{base} = x))$ and hence

$$P(\mathtt{base}) :\equiv \mathbb{Z} \to (\mathtt{base} = \mathtt{base})$$

We seek $\Box : \Sigma_{b:P(\mathtt{base})} ((\mathtt{transport}^P\, \mathtt{loop}\, b) =_{P(\mathtt{base})} b)$, i.e., we seek $\Box_1 : P(\mathtt{base})$ and $\Box_2 : ((\mathtt{transport}^P\, \mathtt{loop}\, \Box_1) =_{P(\mathtt{base})} \Box_1)$. We take

$$\Box_1 :\equiv \lambda z.\mathtt{loop}^z : \mathbb{Z} \to (\mathtt{base} = \mathtt{base})$$

and hence we will have

$$\mathtt{decode}\, \mathtt{base} \equiv \lambda z.\mathtt{loop}^z$$

For defining $\Box_2$, we must analyse transport relative to the family $\lambda x. (\mathtt{Eq}_{S^1}(x) \to (\mathtt{base} = x))$.

Completing the definition of decode

From exercise 6, we get (for $P :\equiv \lambda x.(\mathtt{Eq}_{S^1}(x) \to (\mathtt{base} = x)))$:

$$\mathtt{transport}^{P}\, p\, f\, z = \mathtt{transport}^{\lambda x.(\mathtt{base}=x)}\, p\, f(\mathtt{transport}^{\mathtt{Eq}_{S^1}}\, p^{-1}\, z)$$

which we instantiate with $p :\equiv \mathtt{loop}$ and $f :\equiv \lambda z.\mathtt{loop}^{z}$.

- By a variant of the above property $(*)$, we have

$$(\mathtt{transport}^{\mathtt{Eq}_{S^1}}\, \mathtt{loop}^{-1}\, z) = z - 1$$

Therefore we have

$$\begin{aligned}
&\mathtt{transport}^{P}\, \mathtt{loop}\, (\lambda z.\mathtt{loop}^{z})\, z \\
&\quad = \mathtt{transport}^{\lambda x.(\mathtt{base}=x)}\, \mathtt{loop}\, ((\lambda z.\mathtt{loop}^{z})(z-1)) \\
&\quad \equiv \mathtt{transport}^{\lambda x.(\mathtt{base}=x)}\, \mathtt{loop}\, \mathtt{loop}^{z-1} \\
&\quad = \mathtt{loop}^{z-1} \cdot \mathtt{loop} \\
&\quad = \mathtt{loop}^{z}
\end{aligned}$$

Therefore, by function extensionality, we have proved

$$\mathtt{transport}^{P}\, \mathtt{loop}\, (\lambda z.\mathtt{loop}^{z}) = \lambda z.\mathtt{loop}^{z}$$

and this proof is our $\square_2$.

$(\texttt{encode}\,x)$ and $(\texttt{decode}\,x)$ are quasi-inverses

We prove $\texttt{decode}\,x\,(\texttt{encode}\,x\,p) = p$ by path induction, i.e., we consider only the case where $x \equiv \texttt{base}$ and $p = \texttt{refl}(base)$. Then we have by definition

$$(\texttt{encode}\,\texttt{base}\,\texttt{refl}(\texttt{base})) \equiv 0$$

and in turn

$$\texttt{decode}\,\texttt{base}\,0 \equiv (\lambda z.\texttt{loop}^z)\,0 \equiv \texttt{loop}^0 \equiv \texttt{refl}(\texttt{base})$$

The other direction

• We prove $\mathtt{encode}\, x\, (\mathtt{decode}\, x\, q) = q$ for $q : \mathtt{Eq}_{S^1}(x)$ by induction on S^1, relative to the family

$$P \equiv \lambda x.\, \Pi_{q:\mathtt{Eq}_{S^1}(x)}\, (\mathtt{encode}\, x\, (\mathtt{decode}\, x\, q) = q)$$

We shall need two results:

$$\begin{array}{ll} (**) & \mathbb{Z} \text{ is a set} \\ (***) & \mathtt{encode}\,\mathtt{base}\,(\mathtt{loop}^z) = z \quad (\text{for all } z : \mathbb{Z}) \end{array}$$

We have to provide an inhabitant of

$$\Sigma_{b:P(\mathtt{base})}\, ((\mathtt{transport}^P\, \mathtt{loop}\, b) =_{P(\mathtt{base})} b)$$

But, noticing that $P(\mathtt{base}) \equiv \Pi_{z:\mathbb{Z}}\, (\mathtt{encode}\,\mathtt{base}\,(\mathtt{decode}\,\mathtt{base}\, z) = z)$, we see by $(**)$ that we shall get

$$(\mathtt{transport}^P\, \mathtt{loop}\, b\, z) =_{(\mathtt{encode}\,\mathtt{base}\,(\mathtt{decode}\,\mathtt{base}\, z) = z)} (b\, z)$$

for free, hence by function extensionality we only have to care about producing an inhabitant of $P(\mathtt{base})$, which is given by $(***)$, since by definition

$$(\mathtt{decode}\,\mathtt{base}\, z) \equiv \mathtt{loop}^z$$

Proving $(**)$

The proof that $\mathbb{Z}$ is a set is a variation on the proof that $\mathbb{N}$ is a set, which itself is established via the encode-decode method (see Exercise 16), in a way very much similar to the analysis of $=_{A+B}$ that we have seen.

Proving $(***)$

$(***) \quad \texttt{encode}\,\texttt{base}\,(\texttt{loop}^z) = z \quad$ (for all $z : \mathbb{Z}$)

We first prove

$$(\texttt{encode}\, x\, p) = (\texttt{transport}^{\texttt{Eq}_{S_1}}\, p\, 0)$$

It follows by path induction, since

$$(\texttt{encode}\,\texttt{base}\,\texttt{refl}(\texttt{base})) \equiv 0 \equiv (\texttt{transport}^{\texttt{Eq}_{S_1}}\,\texttt{refl}(\texttt{base})\, 0)$$

Therefore we have

$$(\texttt{encode}\,\texttt{base}\,(\texttt{loop}^z)) = (\texttt{transport}^{\texttt{Eq}_{S_1}}\,\texttt{loop}^z\, 0)$$

Finally, one proves

$$(\texttt{transport}^{\texttt{Eq}_{S_1}}\,\texttt{loop}^z\, 0) = z$$

by induction on $z : \mathbb{Z}$, using

$(*) \quad \texttt{transport}^{\texttt{Eq}_{S^1}}\,\texttt{loop}\, z = z + 1$

Suspension

Let X be a type. The following inductive type captures the notion of suspension of X:

- If $X : \mathcal{U}$, then $\Sigma X : \mathcal{U}$
- Constructors:

$$\begin{aligned} &\mathtt{N} : \Sigma X, && (\text{North}) \\ &\mathtt{S} : \Sigma X && (\text{South}) \\ &\mathtt{merid} : \Pi_{x:X}\, \mathtt{N} =_{\Sigma X} \mathtt{S} && \end{aligned}$$

- Induction operator:

$$\mathtt{ind}_{\Sigma X} : \\ \Pi_{C:\Sigma X \to \mathcal{U}}\, (\Sigma_{n:C\mathtt{N}} \Sigma_{s:C\mathtt{S}} (\Pi_{x:X}\, \mathtt{transport}^{C}\, (\mathtt{merid}\, x)\, n =_{C\mathtt{S}} s) \to (\Pi_{x:\Sigma X}\, Cx)$$

- Associated equalities:

$$\begin{aligned} &\mathtt{ind}_{\Sigma X}\, C\, (n, s, f)\, \mathtt{N} \equiv n && (\text{definitional}) \\ &\mathtt{ind}_{\Sigma X}\, C\, (n, s, f)\, \mathtt{S} \equiv s && (\text{definitional}) \\ &\mathtt{apd}\, (\mathtt{ind}_{\Sigma X}\, C\, (n, s, f))\, (\mathtt{merid}_x) = fx && (\text{propositional}) \end{aligned}$$

Pointed types

• We consider the type $\mathcal{U}_\star$ whose inhabitants are pairs (X, x), where $X : \mathcal{U}$ and $x : X$ (a distinguished point of X). Formally, we consider the type family $\mathtt{id}_{\mathcal{U}} : \mathcal{U} \to \mathcal{U}$, and we define

$$\mathcal{U}_\star :\equiv \Sigma_{X:\mathcal{U}} (\mathtt{id}_{\mathcal{U}}\, X) \equiv \Sigma_{X:\mathcal{U}}\, X$$

We usually write an inhabitant of $\mathcal{U}_\star$ as $(X, \star_X)$ and refer sloppily to X as pointed type.

• Morphisms of pointed types should preserve distinguished points. Given two pointed types $(X, \star_X)$ and $(Y, \star_Y)$, we define a new pointed type $(\mathtt{Map}_\star(X, Y), \star_{XY})$ as follows:

$$\begin{aligned} &\mathtt{Map}_\star(X, Y) :\equiv \Sigma_{f:X\to Y} (f(\star_X) =_Y \star_Y) \\ &\star_{XY} :\equiv \lambda x. \star_Y \end{aligned}$$

We usually write an inhabitant of $\mathtt{Map}_\star(X, Y)$ as $(f, \star_f))$. We refer to f as a pointed map and use the notation $f : X \to_\star Y$ or $X \xrightarrow{f}_\star Y$.

Examples of pointed types

Two "king" examples of pointed types are suspensions and loop spaces. Given a pointed type $(A, \star_A)$:

- we take (by convention) $\mathtt{N}$ as the distinguished point of ΣA;
- we define the loop space $\Omega(A, \star_A)$ of $(A, \star_A)$ as the pointed type $((\star_A =_A \star_A), (\mathtt{refl}(\star_A)))$. We often write ΩA for short.

One could be tempted to choose the distinguished point of X as distinguished point of ΣX, but in this axiomatisation, there is no point in ΣX corresponding to a point in X (only a path "going through it").

Relating suspensions and loop spaces (statement)

We have the following adjunction:

$$\mathtt{Map}_\star((\Sigma A, \mathtt{N}), (B, \star_B)) \overset{\sim}{\longleftrightarrow} \mathtt{Map}_\star((A, \star_A), \Omega(B, \star_B))$$

We shall prove

$$\begin{aligned}\mathtt{Map}_\star((\Sigma A, \mathtt{N}), (B, \star_B)) &\overset{\sim}{\longleftrightarrow} \Sigma_{b_\mathtt{S} : B}(A \to (\star_B = b_\mathtt{S}))\\ \mathtt{Map}_\star((A, \star_A), \Omega(B, \star_B)) &\overset{\sim}{\longleftrightarrow} \Sigma_{b_\mathtt{S} : B}(A \to (\star_B = b_\mathtt{S}))\end{aligned}$$

We shall use two properties proved before, which we recall. Let $P : A \to \mathcal{U}$ be a type family.

(i) If each $P(x)$ is contractible, then $\Sigma_{x:A} P(x)$ is equivalent to A.

(ii) If A is contractible with center a_0, then $\Sigma_{x:A} P(x)$ is equivalent to $P(a_0)$.

Relating suspensions and loop spaces (proof)

• An inhabitant of $\mathtt{Map}_\star((\Sigma A, \mathtt{N}), (B, \star_B))$ is given by two elements $b_\mathtt{N}$, $b_\mathtt{S}$, a function $h_1 : A \to (b_\mathtt{N} = b_\mathtt{S})$ and a path from $b_\mathtt{N}$ to $\star_B$. This can be repackaged as

$$\Sigma_{p:\Sigma_{b_\mathtt{N}:B}(b_\mathtt{N}=\star_B)} \Sigma_{b_\mathtt{S}:B}(A \to ((\mathtt{pr}_1 p) = b_\mathtt{S}))$$

which isolates the contractible type $\Sigma_{b_\mathtt{N}:B}(b_\mathtt{N} = \star_B)$. Hence by property (ii) above, we have proved $\mathtt{Map}_\star((\Sigma A, \mathtt{N}), (B, \star_B)) \cong \Sigma_{b_\mathtt{S}:B}(A \to (\star_B = b_\mathtt{S}))$.

• An inhabitant of $\mathtt{Map}_\star((A, \star_A), \Omega(B, \star_B))$ is a function $g : A \to (\star_B = \star_B)$, together with a path from $(g\,\star_A)$ to $\mathtt{refl}(\star_B)$. So we are left to prove

$$\Sigma_{g:A\to(\star_B=\star_B)}((g\,\star_A) = \mathtt{refl}(\star_B)) \cong \Sigma_{b_\mathtt{S}:B}(A \to (\star_B = b_\mathtt{S}))$$

We have (using (ii) and (i), respectively):

$$\begin{array}{l}\Sigma_{g:A\to(\star_B=\star_B)}((g\,\star_A) = \mathtt{refl}(\star_B)) \cong \Sigma_{r:\Sigma_{b_\mathtt{S}:B}(\star_B=b_\mathtt{S})} \Sigma_{g:A\to(\star_B=(\mathtt{pr}_1\,r))}((g\,\star_A) = (\mathtt{pr}_2 r)) \\ \Sigma_{b_\mathtt{S}:B}(A \to (\star_B = b_\mathtt{S})) \cong \Sigma_{b_\mathtt{S}:B}\Sigma_{g:A\to(\star_B=b_\mathtt{S})}\Sigma_{q:(\star_B=b_\mathtt{S})}((g\,\star_A) = q)\end{array}$$

and conclude since both right hand sides are an easy repackaging of each other.

(Note the clever use of the sequence $b_\mathtt{S} : B$, $q : (\star_B = b_\mathtt{S})$ and $((g\,\star_A) = q)$, giving rise to the two contractible types $\Sigma_{b_\mathtt{S}:B}(\star_B = b_\mathtt{S})$ and $\Sigma_{q:(\star_B=b_\mathtt{S})}((g\,\star_A) = q)$.)

Fiber sequences

Given pointed types $X, Y, Z : \mathcal{U}_\star$, we say that two pointed maps

$$X \xrightarrow{f}_\star Y \xrightarrow{g}_\star Z$$

form a (short) fiber sequence if, for all $y : Y$ we have a map

$$\epsilon_y : \mathtt{fib}_f(y) \to (g(y) = \star_Z)$$

which is a quasi-inverse. Morevoer, we require $\epsilon_{\star_Y}$ to be pointed, i.e.,

$$\epsilon_{\star_Y}(\star_X, \star_f) = \star_g$$

(note that for general y, the types $\mathtt{fib}_f(y)$ and $(g(y) = \star_Z)$ need not be inhabited).
In the following, we shall state the results on fiber sequences without proof. We refer to the HoTT book.

Fiber sequences versus exact sequences

• In homological algebra, a sequence s $M \xrightarrow{f} N \xrightarrow{g} P$ of linear maps (or module morphisms) is called exact if $\mathtt{im}(f) = \mathtt{ker}(g)$, i.e., if for all $y \in N$ we have $g(y) = 0$ if and only if $y = f(x)$ for some $x \in M$.

• In group theory, a sequence $G \xrightarrow{f} H \xrightarrow{g} K$ of group morphisms is called exact if for $y \in H$ we have $h(y) = e$ if and only if $y = f(x)$ for some $x \in G$.

If we look at the definition of fiber sequence in a proof-irrelevant way, and think of $\mathtt{fib}_f(y)$ and $(g(y) = \star_Z)$ as propositions, then the notion of fiber sequence boils down to saying that for all $y : Y$ we have $g(y) = \star_Z$ if and only if $f(x) = y$ for some x. But precisely because in general those two types will have higher structure, the notion of fiber sequence is much richer.

Canonical fiber sequence associated with a pointed map

Given pointed types $X, Y : \mathcal{U}_\star$ and $f : X \to_\star Y$, the sequence

$$\mathtt{fib}_f(\star_Y) \overset{\mathtt{pr}_1}{\to} X \overset{f}{\to}_\star Y$$

is a fiber sequence.

We note that $\mathtt{pr}_1$ is pointed, since $\mathtt{pr}_1(\star_X, \star_f) \equiv \star_X$.

Every fiber sequence is of this sort, in the following sense: if $Z \overset{g}{\to}_\star X \overset{f}{\to}_\star Y$ is a fiber sequence, then we have

$$(Z, g) =_{\Sigma V : \mathcal{U}_\star .,\mathtt{Map}_\star(V,X)} (\mathtt{fib}_f(\star_Y), \mathtt{pr}_1)$$

(here, $\Sigma V : \mathcal{U}_\star \, \mathtt{Map}_\star(V, X)$ is the pointed version of $\mathcal{U}/Y$).

Long fiber sequence associated with a pointed map

Our data are still $X, Y : \mathcal{U}_\star$ and $f : X \to_\star Y$. We can iterate the construction $\mathtt{fib}_f(\star_Y) \overset{\mathtt{pr}_1}{\to} X \overset{f}{\to}_\star Y$.
We set

$$X^{(0)} :\equiv Y \quad \star_{(0)} :\equiv \star_Y \qquad X^{(1)} :\equiv X \quad \star_{(1)} :\equiv \star_X \qquad f^{(0)} :\equiv f$$

and for $n \geq 1$, we set

$$\begin{aligned} & f^{(n)} :\equiv \mathtt{pr}_1 \\ & X^{(n+1)} :\equiv \mathtt{fib}_{f^{(n-1)}}(\star_{(n-1)}) \\ & \star_{(n+1)} :\equiv (\star_{(n)}, \star_{f^{(n-1)}}) \end{aligned}$$

In this way, we obtain a long fiber sequence

$$\ldots \overset{f^{(n+1)}}{\to}_\star X^{(n+1)} \overset{f^{(n)}}{\to}_\star X^{(n)} \overset{f^{(n-1)}}{\to}_\star \ldots \to_\star X^{(2)} \overset{f^{(1)}}{\to}_\star X^{(1)} \overset{f^{(0)}}{\to}_\star X^{(0)}$$

meaning that for all n, the short sequence

$$X^{(n+2)} \overset{f^{(n+1)}}{\to}_\star X^{(n+1)} \overset{f^{(n)}}{\to}_\star X^{(n)}$$

is a short fiber sequence.

Reading the long fiber sequence in terms of loop spaces

Consider the last five pointed types in the long sequence:

$$X^{(4)} \overset{f^{(3)}}{\to}_{\star} X^{(3)} \overset{f^{(2)}}{\to}_{\star} X^{(2)} \overset{f^{(1)}}{\to}_{\star} X \overset{f}{\to}_{\star} Y$$

Then we have

$$X^{(3)} \overset{\sim}{\longleftrightarrow} \Omega Y \qquad X^{(4)} \overset{\sim}{\longleftrightarrow} \Omega X$$

with pointed quasi-inverses. Moreover, under these equivalences, $f^{(3)}$ is identified with $\Omega f : \Omega X \to \Omega Y$, defined as follows, for all $p : \star_X = \star_X$:

$$\Omega(f)(p) :\equiv (\star_f)^{-1} \cdot f(p) \cdot \star_f$$

Setting $Z :\equiv X^{(2)}$, we thus have a long fiber sequence (known to topologists)

$$\cdots \to \Omega^2 X \to \Omega^2 Y \to \Omega Z \to \Omega X \to \Omega Y \to Z \to X \to Y$$

The long exact sequence of homotopy groups

There is a more down-to-earth even and more anciently well-known long sequence in topology: with every exact $Z \xrightarrow{f}_\star X \xrightarrow{g}_\star Y$ is associated an exact sequence of groups:

$$\cdots \to \pi_3(Y) \to \pi_2(Z) \to \pi_2(X) \to \pi_2(Y) \to \pi_1(Z) \to \pi_1(X) \to \pi_1(Y)$$

where $\pi_n(A)$ is the n-th homotopy group of the based space A (classes of maps from S^n to A up to homotopy).

Type theory as described so far lacks something to account for this! We need one more ingredient: truncation. We need a way to say in type theory that we want to force a type to be a proposition, or a set (as here: homotopy groups are sets, sets with structure, but sets, not fancy higher groupoids).

Truncation

There are truncations for each h-level n

- If $A : \mathcal{U}$, then $\|A\|_n : \mathcal{U}$.

- Constructors: we have a function $|\ | : A \to \|A\|_n$.

- If $P : A \to U$ is a family of n-types and if $f : \Pi_{x:A} Px$ then we have a function $\tilde{f} : \Pi_{x:\|A\|_n} Px$.

- Definitional equality: $\tilde{f}(|x|) \equiv f(x)$ (for all $x : A$).

Then we can define the homotopy groups in type theory by

$$\pi_n(X, \star_X) = \|\Omega^n(X, \star_X)\|_0$$

and then we can go on and establish the long exact sequence of homotopy groups, deriving it from the long fiber sequence of spaces.

THE END

Epilogue

The more traditional definition of $\pi_n(X)$ for a pointed space is as the set of all pointed maps from S^n to X (quotiented by homotopy). Here is how it connects to $\Omega^n(X, \star_X)$ (using the type-theoretical language), informally.

- The data of a pointed map from S^1 to $(X, \star_X)$ amounts, by recursion, to give a point $a : X$ and a path $p : (a = a)$. But since the map is pointed, and since it maps definitionally `base` to a, this forces $a = \star_X$. Hence we can identify $S^1 \to_\star X$ with ΩX.
- Suppose that we have established

$$\mathtt{Map}_\star(S^{n-1}, X) \overset{\sim}{\longleftrightarrow} \Omega^{n-1} Y \quad \text{(for all } Y\text{)}$$

  Then, by the adjunction between Σ and Ω and by the property that S^n is the suspension of S^{n-1} (which serves as type-theoretic definition of the spheres S^n for $n > 1$), we have

$$\begin{aligned} \mathtt{Map}_\star(S^n, X) \;\equiv\; \mathtt{Map}_\star(\Sigma S^{n-1}, X) \;&\overset{\sim}{\longleftrightarrow} \mathtt{Map}_\star(S^{n-1}, \Omega X) \\ &\overset{\sim}{\longleftrightarrow} \Omega^{n-1}(\Omega X) \qquad \equiv \;\; \Omega^n X \end{aligned}$$

Exercises 1-3

In the intuitionistic setting, negation is defined as $A \to 0$.

Exercise 1: Let $A : \mathcal{U}$ and $B : \mathcal{U}$. Construct a term (a de Morgan law!) of type $((A + B) \to 0) \to (A \to 0) \times (B \to 0)$.

Exercise 2: Let $A : \mathcal{U}$, $P : A \to \mathcal{U}$ and $Q : A \to \mathcal{U}$. Construct a term of type $(\Pi_{x:A}(Px \times Qx)) \to ((\Pi_{x:A}Px) \times (\Pi_{x:A}Qx))$.

Exercise 3 We define $\mathtt{Can} : \mathbb{N} \to \mathcal{U}$ by induction (notice the typographical difference: on the right, $\mathtt{0}$ and $\mathtt{1}$ are types!) :

$$\mathtt{Can}\, 0 :\equiv \mathtt{0} \qquad \mathtt{Can}\, (\mathtt{succ}\, n) :\equiv (\mathtt{Can}\, n) + \mathtt{1}$$

We then define $\mathtt{Fin}\, n :\equiv \Sigma_{A:\mathcal{U}}(A =_{\mathcal{U}} \mathtt{Can}\, n)$. Define the function $\mathtt{max}$ and $i_n = \mathtt{pick}\, i\, n\, p$, where $p : i < n$, where $(x < y) :\equiv ((\mathtt{succ}\, x) \leq y)$, and where $\leq : \Pi_{x,y:\mathbb{N}}\mathcal{U}$ is defined by induction on x, and then on y:

$$(0 \leq y) :\equiv \mathtt{1} \qquad (\mathtt{succ}\, x \leq 0) :\equiv \mathtt{0} \qquad (\mathtt{succ}\, x \leq \mathtt{succ}\, y) :\equiv (x \leq y)$$

Exercises 4-6

Exercise 4 One can encore coproducts from booleans (using a Σ-type):

$$A + B = \Sigma_{x:2}\mathtt{rec}_2\,\mathcal{U}\,A\,B\,x$$

Define $\mathtt{rec}_{A+B}$ using $\mathtt{rec}_2$.

Exercise 5 One can encode products from booleans (using a Π-type):

$$A \times B :\equiv \Pi_{x:2}\mathtt{rec}_2\,\mathcal{U}\,A\,B\,x$$

Define (a, b), $\mathtt{pr}_1$, $\mathtt{pr}_2$ and their definitional and propositional equalities.

Exercise 6 Consider two families $P : A \to \mathcal{U}$ and $Q : A \to \mathcal{U}$, and let

$$R\,x := (P\,x) \to (Q\,x)$$

Show that, for $x, y : A$ and $p : (x =_A y)$, and $f : (P\,x) \to (Q\,x)$, $u : (P\,y)$, we have:

$$\mathtt{transport}^R\,p\,f\,u = \mathtt{transport}^Q\,p\,f(\mathtt{transport}^P\,p^{-1}\,u)$$

Exercises 7-11

Exercise 7 Let $P : A \to \mathcal{U}$ and $Q : (\Sigma_{x:A} P\,x) \to \mathcal{U}$. These data generate a family $R : A \to \mathcal{U}$ of Σ-types, defined by

$$R\,x :\equiv \Sigma_{u:P\,x} Q\,(x,u)$$

Let $p : (x =_A y)$ and $(u,z) : R\,x$. Find a formula for $\mathtt{transport}^R$ in terms of $\mathtt{transport}^P$ and $\mathtt{transport}^Q$.

Exercise 8 Show that if B is a proposition, then, for any type $A : \mathcal{U}$, $A \to B$ is a proposition (hint: use function extensionality).

Exercise 9 Let $f, g : A \to B$, $H : f \sim g$ and $p : x =_A y$. Show that (naturality)

$$H(x) \cdot g(p) =_{f(x) =_B g(y)} f(p) \cdot H(y)$$

Exercise 10. Let $f : A \to A$ and $H : f \sim \mathtt{id}_A$. Show that

$$H(f(x)) =_{f(f(x)) =_A f(x)} f(H(x))$$

Exercise 11 Show that for any A and $a : A$, the type $\Sigma_{x:A}\, a =_A x$ is contractible.

Exercises 12-16

Exercise 12 Show that if A is contractible, then so are all $x =_A y$ for $x, y : A$.

Exercise 13 Let $B : A \to \mathcal{U}$. Show that if all Bx are contractible and if A is contractible, then $\Sigma_{x:A} Bx$ is contractible.

Exercise 14 Let $A : \mathcal{U}$, $B : A \to \mathcal{U}$, and $C : \Sigma_{x:A} \to \mathcal{U}$. Show that

$$(\Sigma_{x:A}\Sigma_{y:Bx} C(x,y)) \overset{\sim}{\longleftrightarrow} (\Sigma_{z:\Sigma_{x:A}Bx} Cz)$$

Exercise 15 Give formal definitions of $\lambda z.z + 1 : \mathbb{Z} \to \mathbb{Z}$ and $\lambda z.\,\mathtt{loop}^z$.

Exercise 16 Show that $\mathbb{N}$ is a set.

Solution to exercise 3

$$\mathtt{max}\,0 :\equiv (\mathtt{inr}\,\star) \qquad \mathtt{max}\,(\mathtt{succ}\,n) :\equiv (\mathtt{inl}\,(\mathtt{max}\,n))$$

and $\mathtt{pick} :\ \Pi_{i:\mathbb{N}}\, Pi_{i:\mathbb{N}}\,(i < n) \to \mathtt{Can}\,(n)$ is defined by

$$\begin{aligned}
&\mathtt{pick}\,i\,0\,p :\equiv (\mathtt{rec}_0\,\mathbb{N}\,p)\\
&\mathtt{pick}\,0\,\mathtt{succ}(n)\,p :\equiv (\mathtt{inr}\,\star)\\
&\mathtt{pick}\,(\mathtt{succ}\,i)\,(\mathtt{succ}\,n)\,p :\equiv \mathtt{inl}(\mathtt{pick}\,i\,n\,p)
\end{aligned}$$

Solutions to exercises 4-6

(Exercise 4) We derive, for $f : A \to C$, $g : B \to C$,

$$\mathtt{rec}_{A+B}\, C\, f\, g :\equiv \mathtt{rec}_{\Sigma_{x:2}\mathtt{rec}_2 \mathcal{U}\, A\, B\, x}\, C\, \Box : \Sigma_{x:2}\mathtt{rec}_2 \mathcal{U}\, A\, B\, x \to C$$

where $\Box : \Pi_{x:2} Dx$, with $D = \lambda x.(\mathtt{rec}_2 \mathcal{U}\, A\, B\, x \to C)$. We can fill the unknown $\Box$ with $\mathtt{ind}_2\, D\, f\, g$, noticing that $D 0_2 \equiv A \to C$ and $D 1_2 \equiv B \to C$. Summing up:

$$\mathtt{rec}_{A+B}\, C\, f\, g :\equiv \mathtt{rec}_{\Sigma_{x:2}\mathtt{rec}_2 \mathcal{U}\, A\, B\, x}\, C\, (\mathtt{ind}_2\, D\, f\, g)$$

(Exercise 5) $(a, b) :\equiv \mathtt{ind}_2\, (\mathtt{rec}_2 \mathcal{U}\, A\, B)\, a\, b$, $\mathtt{pr}_1\, c :\equiv c\, 0_2$, $\mathtt{pr}_2\, c :\equiv c\, 1_2$. We have $\mathtt{pr}_1\, (a, b) :\equiv \mathtt{ind}_2\, (\mathtt{rec}_2 \mathcal{U}\, A\, B)\, a\, b\, 0_2 \equiv a$. We can also derive surjective pairing. But this requires function extensionality.

(Exercise 6): The formula type-checks, because

- $\mathtt{transport}^P\, p^{-1}\, u : (P\, x)$
- $f(\mathtt{transport}^P\, p^{-1}\, u) : (Q\, x)$
- $\mathtt{transport}^Q\, p\, f(\mathtt{transport}^P\, p^{-1}\, u) : (Q\, y)$

The proof of inhabitation is immediate by path induction.

Solution to Exercise 7

We seek $\mathtt{transport}^R\, p\,(u, z) = (\square_1, \square_2) : R\, y$, hence we must have $\square_1 : P\, y$ and $\square_2 : Q\,(y, \square_1)$. We take

$$\begin{aligned}\square_1 &:\equiv \mathtt{transport}^P\, p\, u\\ \square_2 &:\equiv \mathtt{transport}^Q\, ((\mathtt{pair}^=)\,(p, (\mathtt{refl}\,(\mathtt{transport}^P\, p\, u))))\, z\end{aligned}$$

Indeed, we have $z : Q\,(x, u)$ and $(\mathtt{pair}^=)\,(p, (\mathtt{refl}\,(\mathtt{transport}^P\, p\, u))) : (x, u) = (y, (\mathtt{transport}^P\, p\, u))$.

Solution to Exercises 12-13

Exercise 12: Let $(a, f) : \mathtt{isContr}(A)$. Our goal is to prove $\mathtt{isContr}(x =_A y)$ (for x, y arbitrary). We fix x and let y vary. We can take

$$h\,x\,y :\equiv (fx)^{-1} \cdot fy$$

as center. We look for $H_x : \Pi_{y:A}\Pi_{p:x=_A y}.C\,y\,p$, where $C\,y\,p :\equiv (h\,x\,y =_{x=_A y} p)$. By based path induction on A, it is enough to prove

$$h\,x\,x =_{x=_A x} \mathtt{refl}(x),$$

which holds since $h\,x\,x \equiv (fx)^{-1} \cdot fx = \mathtt{refl}(x)$.

Exercise 13 We have shown that if all Bx are contractible, the $\Sigma_{x:A}Bx$ is equivalent to A. Therefore, if A is contractible, then so is $\Sigma_{x:A}Bx$.

Sketch of solution to Exercise 16

The idea is to use the encode-decode method to characterise all types $m =_{\mathbb{N}} n$. One defines (cf. also the inequality predicates of Exercise 3):

$$\begin{aligned}
&\mathtt{code}(0,0) :\equiv \mathtt{1}\\
&\mathtt{code}(\{ \mathit{ttsucc}(n), 0) :\equiv \mathtt{0}\\
&\mathtt{code}(0, \mathrm{succ}(n)) :\equiv \mathtt{0}\\
&\mathtt{code}(\mathrm{succ}(m), \mathrm{succ}(n)) :\equiv \mathtt{code}(m,n)
\end{aligned}$$

An then to prove

$$(m =_{\mathbb{N}} n) \overset{\sim}{\longleftrightarrow} \mathtt{code}(m,n)$$

by a suitable combination of induction on $\mathbb{N}$ and of path induction. The conclusion then follows from the fact that by definition all the types $\mathtt{code}(m,n)$ are either $\mathtt{0}$ or $\mathtt{1}$, which are propositions.