Topic 7: Using identity types

June 10, 2014

Now we would like to learn how to use identity types and how to do some actual mathematics with them. By now we have essentially introduced all inference rules of HoTT; the main missing ingredient is the univalence axiom. At this stage it is possible to do a substantial amount of mathematics, and some of this we will do here.

Throughout $A, B : \mathcal{U}$ are fixed types.

Example: Propositional uniqueness principles

Recall the *uniqueness principle* for functions, which is an inference rule stating that $f \equiv \lambda x.f(x)$ for any (dependent) function f. For other types, we did not have such uniqueness principles; for example for $u : \mathbf{1}$, such a uniqueness principle would assert that $u \equiv \star$, but we have not postulated this and it is not derivable. However, at this point we can show that such a uniqueness principle holds with propositional equality in place of judgemental equality:

Theorem 1. $\prod_{u:\mathbf{1}} u = \star$.

The proof of this is exemplative of many other proofs in HoTT. It is a good example of how one can derive non-trivial equations in HoTT, although the only way to prove an equation is to reduce it in some way to reflexivity.

Proof. In order to find an element $w : \prod_{u:\mathbf{1}} u = \star$, we use induction on u. This reduces the problem to finding $w(\star) : \star = \star$, which we can define to be $w(\star) :\equiv \mathrm{refl}_\star$. □

The same theorem and proof in Coq:

```
Theorem prop_unique : forall u : unit, u = tt.
Proof.
        intro u.
        induction u.
        reflexivity.
Defined.
```

Analogous propositional uniqueness principles can now be derived for many other types as well:

Theorem 2. $\prod_{x:A\times B} x = (\mathrm{pr}_1(x), \mathrm{pr}_2(x))$.

Proof. In order to find an element of the type $\prod_{x:A\times B} x = (\mathrm{pr}_1(x), \mathrm{pr}_2(x))$, we use induction on x. This reduces the problem to finding an element

$$v : \prod_{a:A} \prod_{b:B} (a, b) = (\mathrm{pr}_1((a, b)), \mathrm{pr}_2((a, b))).$$

Since $\mathrm{pr}_1((a,b)) \equiv a$ and $\mathrm{pr}_2((a,b)) \equiv b$, we can take $v :\equiv \lambda a.\lambda b.\mathrm{refl}_{(a,b)}$. $\square$

```
Theorem prop_unique_product (A B : Type) : forall x : A * B, x = (fst x, snd x).
Proof.
        intro x.
        induction x.
        simpl.
        reflexivity.
Defined.
```

One can generalize this to the dependent pair type: for any dependent type $P : A \to \mathcal{U}$, one can similarly derive $x = (\mathrm{pr}_1(x), \mathrm{pr}_2(x))$ for any $x : \sum_{a:A} P(a)$. We will prove a more general statement later on, when we characterize the identity type $p = q$ for $p, q : \sum_{a:A} P(a)$.

Functions acting on paths

If we have $x, y : A$ together with $p : x = y$ and a function $f : A \to B$, does this yield an element of $f(x) = f(y)$? In other words, does function application preserve propositional equality? In terms of the topological interpretation, one can say that we now analyse how functions act on paths.

Lemma 3. *For any function $f : A \to B$ and $x, y : A$, there is a function*

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

defined such that for all $x : A$, $\mathrm{ap}_f(\mathrm{refl}_x) \equiv \mathrm{refl}_{f(x)}$.

Strictly speaking, we also need to consider the two elements $x, y : A$ as arguments of the function ap_f, whose type therefore should be

$$\mathrm{ap}_f : \prod_{x,y:A} (x =_A y) \to (f(x) =_B f(y)).$$

But since the first two arguments x and y are determined by the third argument $p : x = y$, we consider x and y to be "implicit" arguments which are not written down explicitly, except for in the following proof.

Proof. For any $x, y : A$ and any $p : x =_A y$, we want to find $\mathrm{ap}_f(x, y, p) : f(x) =_B f(y)$. By induction on p as an element of the (say, unbased) identity type, it is sufficient to do so in the case that $y \equiv x$ and $p \equiv \mathrm{refl}_x$. But in this case, we can put $\mathrm{ap}_f(x, x, \mathrm{refl}_x) :\equiv \mathrm{refl}_{f(x)} : f(x) =_B f(x)$. $\square$

In `Coq`, the statement and proof look as follows:

```
Lemma ap {A B : Type} (f : A → B) : forall x y : A, (x = y) → (f(x) = f(y)).
Proof.
        intros x y p.
        induction p.
        reflexivity.
Defined.
```

We interpret Lemma 3 as saying that *every* function f is automatically continuous, since it maps paths to paths! So in HoTT, it is impossible to construct a function that is not continuous.

Example: commutativity of addition

As the possibly first example of actual mathematics in HoTT, let us revisit the natural numbers and prove that addition is commutative. The following proof structure coincides exactly with how one would prove commutativity of addition in conventional foundations! This is true for many other things as well.

Lemma 4. $\prod_{x,y:\mathbb{N}} \mathtt{succ}(y+x) = y + \mathtt{succ}(x)$.

Recall that we had defined addition $x+y :\equiv \mathtt{add}(x,y)$ by induction over the first argument, using the definitions $\mathtt{add}(0,y) :\equiv y$ and $\mathtt{add}(\mathtt{succ}(x),y) :\equiv \mathtt{succ}(\mathtt{add}(x,y))$.

Proof. In order to find an element of the type $\mathtt{succ}(y+x) = y + \mathtt{succ}(x)$ for any $x, y : \mathbb{N}$, we can use induction on y. For $y \equiv 0$, we have $\mathrm{refl}_{\mathtt{succ}(x)} : \mathtt{succ}(x) = \mathtt{succ}(x)$, and by the definition of $+$ we also have judgemental equalities $\mathtt{succ}(0+x) \equiv \mathtt{succ}(x)$ and $0 + \mathtt{succ}(x) \equiv \mathtt{succ}(x)$. Therefore, we also have $\mathrm{refl}_{\mathtt{succ}(x)} : \mathtt{succ}(0+x) = 0 + \mathtt{succ}(x)$.

For the induction step, we need to show that one can construct an element of the identity type $\mathtt{succ}(\mathtt{succ}(y)+x) = \mathtt{succ}(y) + \mathtt{succ}(x)$ from any given element $v : \mathtt{succ}(y+x) = y + \mathtt{succ}(x)$. Since $\mathrm{ap}_{\mathtt{succ}}(v) : \mathtt{succ}(\mathtt{succ}(y+x)) = \mathtt{succ}(y + \mathtt{succ}(x))$ and we have judgemental equalities

$$\mathtt{succ}(\mathtt{succ}(y)+x) \equiv \mathtt{succ}(\mathtt{succ}(y+x)), \qquad \mathtt{succ}(y) + \mathtt{succ}(x) \equiv \mathtt{succ}(y + \mathtt{succ}(x))$$

by the definition of $+$, we also have $\mathrm{ap}_{\mathtt{succ}}(v) : \mathtt{succ}(\mathtt{succ}(y)+x) = \mathtt{succ}(y) + \mathtt{succ}(x)$. □

Let us write $\mathtt{add\_succ} : \prod_{x,y:\mathbb{N}} y + \mathtt{succ}(x) = \mathtt{succ}(y+x)$ for the element constructed in the proof.

The same statement and proof in Coq:

```
Lemma add_succ : forall x y : nat, x + S y = S (x + y).
Proof.
        intros x y.
        induction x.
                simpl.
                reflexivity.

                simpl.
                apply (ap S).
                exact IHx.
Defined.
```

Theorem 5. $\prod_{x,y:\mathbb{N}} x + y = y + x$.

Proof. In order to find an element of the type $x + y = y + x$ for any x and y, we can use induction on x. This reduces the problem to finding elements of the following two types:

- $0 + y = y + 0$,
- $(x + y = y + x) \longrightarrow (\mathtt{succ}(x) + y = y + \mathtt{succ}(x))$

We start with the first. Since $+$ was defined by induction over the first argument, we have $0 + y \equiv y$, so it remains to show that $y = y + 0$. This in turn can be shown by induction on y as follows. For $y \equiv 0$, we have $0 \equiv 0 + 0$ again by definition of $+$, and hence we have a judgemental equality of types

$(0 = 0) \equiv (0 = 0 + 0)$, and therefore $\mathrm{refl}_0 : 0 = 0 + 0$. Concerning the induction step, if we have an equality witness $w : y = y + 0$, then we obtain $\mathrm{ap}_{\mathtt{succ}}(w) : \mathtt{succ}(y) = \mathtt{succ}(y + 0)$. Since $\mathtt{succ}(y) + 0 \equiv \mathtt{succ}(y + 0)$ by the inductive definition of $+$, we also have $\mathrm{ap}_{\mathtt{succ}}(w) : \mathtt{succ}(y) = \mathtt{succ}(y) + 0$.

Now for the main induction step. Suppose that we have $v : x + y = y + x$, and we want to find an element of $\mathtt{succ}(x) + y = y + \mathtt{succ}(x)$. For every y, we have

$$\mathrm{ap}_{\mathtt{succ}}(v) \;:\; \mathtt{succ}(x) + y = \mathtt{succ}(y + x),$$

where we have used $\mathtt{succ}(x + y) \equiv \mathtt{succ}(x) + y$ by the inductive definition of $+$, and

$$\mathtt{add\_succ}(x, y) \;:\; \mathtt{succ}(y + x) = y + \mathtt{succ}(x)$$

thanks to Lemma 4. The claim now follows from transitivity of equality; this is what we get to next. $\square$

Concatenating paths

Last time, we already saw that propositional equality is symmetric, in the sense that there exists a canonical function $(x = y) \to (y = x)$ which maps any path $p : x = y$ to its "inverse" $p^{-1} : y = x$. In the path interpretation, this corresponds to the fact that every path can be traversed forwards or backwards. Recall that taking the inverse is a function

$$\_^{-1} \;:\; \prod_{x,y:A} (x = y) \to (y = x)$$

which can be defined by (based or unbased) path induction, which reduces the problem to defining the function on $y \equiv x$ and refl_x, in which case we define the function value to be refl_x itself. Although x and y (and even A) should formally also be considered as arguments of this function, we omit them for brevity; if one knows p, one can deduce x and y from the type of p, and then A as the type of x and y. To summarize, the function $p \mapsto p^{-1}$ is defined by path induction on p, together with the stipulation that $\mathrm{refl}_x^{-1} :\equiv \mathrm{refl}_x$ for every $x : A$.

Similarly, it is possible to *concatenate* paths: if p is a path from x to y and q is a path from y to z, then there is a concatenated path $p \cdot q$ from x to z. This is indeed the case: for any $A : \mathcal{U}$, one can construct a function

$$\_\cdot\_ \;:\; \prod_{x,y,z:A} (x = y) \to (y = z) \to (x = z) \tag{1}$$

as follows. By permuting the z argument with the $(x = y)$-argument, it is enough to construct this function in the form

$$\_\cdot\_ \;:\; \prod_{x,y} (x = y) \to \prod_{z:A} (y = z) \to (x = z).$$

Applying (unbased) path induction lets us assume that we are dealing with the case $y \equiv x$ and $\mathrm{refl}_x : x = x$, in which case we simply need to construct an element of the type

$$\prod_{z:A} (x = z) \to (x = z),$$

for which we could simply take the identity function for every z. However, it turns out to be better to *not* use the identity function here, but rather to define such a function again through path induction, which allows us to put $z \equiv x$ and assume $\mathsf{refl}_x : x = x$, in which case we take the value of the function to again be $\mathsf{refl}_x : x = x$.

The reason for not using the identity function, but instead using another path induction, is mainly aesthetics: with the identity function, the resulting computation rule for $\cdot$ is,

$$\mathsf{refl}_x \cdot q \equiv q$$

for all $q : y = z$, while with a second path induction, the computation rule is merely

$$\mathsf{refl}_x \cdot \mathsf{refl}_x \equiv \mathsf{refl}_x,$$

which we prefer, since it makes both arguments behave in the same way.

In logical terms, one can interpret (1) as the transitivity of equality: if x is equal to y and y is equal to z, then x is also equal to z. The function (1) can be interpreted as converting any two proofs $p : x = y$ and $q : y = z$ into $p \cdot q : x = z$. Generally speaking, thinking in terms of paths between points in a space rather than proofs of equalities is the preferred intuition, and so we stick with this one from now on.

Theorem 6. *Concatenation of paths is associative: for any $w, x, y, z : A$ and $p : w = x$, $q : x = y$, $r : y = z$,*

$$p \cdot (q \cdot r) = (p \cdot q) \cdot r.$$

Strictly speaking, what me mean by this theorem is that we claim to be able to construct an element of the type

$$\prod_{A:\mathcal{U}} \prod_{w,x,y,z:A} \prod_{p:w=x} \prod_{q:x=y} \prod_{r:y=z} p \cdot (q \cdot r) = (p \cdot q) \cdot r, \tag{2}$$

and this is what will be done in the proof.

Proof. We use similar arguments as in the construction of $\cdot$: permuting some of the parameters w, x, y, z and p, q, r and then applying path induction. Indeed, path induction over p allows us to assume $x \equiv w$ and $p \equiv \mathsf{refl}_w$, in which case we have $q : w = y$ and $r : y = z$ and need to prove

$$\mathsf{refl}_w \cdot (q \cdot r) = (\mathsf{refl}_w \cdot q) \cdot r.$$

We show this by another path induction, this time on q, which gives $y \equiv w$ and $q \equiv \mathsf{refl}_w$, so that we have reduced the problem to finding an element of

$$\mathsf{refl}_w \cdot (\mathsf{refl}_w \cdot r) = (\mathsf{refl}_w \cdot \mathsf{refl}_w) \cdot r.$$

Using the same kind of path induction for r and applying the computation rule $\mathsf{refl}_w \cdot \mathsf{refl}_w \equiv \mathsf{refl}_w$ results in the assertion

$$\mathsf{refl}_w = \mathsf{refl}_w,$$

which has a trivial proof given by $\mathsf{refl}_{\mathsf{refl}_w}$. □

This result is our first *coherence* law. Generally speaking, a coherence law states that any two ways to construct a new object from given data are equal. In this case, it states that the concatenation of a sequence of paths does not depend on the order in which these paths are concatenated—at least up to propositional equality. The element of the above type (2) can be shown to satisfy its own coherence law!

There is also a sense in which the reflexivity elements are unit elements for the concatenation of paths, in the sense that concatenating with them gives the other path back—of course, again up to propositional equality:

Theorem 7. *For any $x, y : A$ and $p : x = y$,*

$$\mathrm{refl}_x \cdot p = p, \qquad p \cdot \mathrm{refl}_y = p.$$

Finally, in the same sense, concatenating a path with its inverse recovers the reflexivity elements:

Theorem 8. *For any $x, y : A$ and $p : x = y$,*

$$p \cdot p^{-1} = \mathrm{refl}_x, \qquad p^{-1} \cdot p = \mathrm{refl}_y.$$

Both theorems can easily be proven by path induction on p.

The concatenation of paths is an algebraic structure on all the identity types over any type A which turns A into a so-called *groupoid*. Can similar things also be done with *higher paths*, i.e. with elements of identity types between paths, identities between those, and so on? Before getting to this, we briefly study how the algebraic structure on paths interacts with the ap function from Lemma 3.

Theorem 9. *The function* ap *has the following properties:*

1. $\mathrm{ap}_f(p \cdot q) = \mathrm{ap}_f(p) \cdot \mathrm{ap}_f(q)$,
2. $\mathrm{ap}_f(p^{-1}) = \mathrm{ap}_f(p)^{-1}$,
3. *For $f : A \to B$ and $g : B \to C$, we have* $\mathrm{ap}_{g \circ f}(p) = \mathrm{ap}_g(\mathrm{ap}_f(p))$.
4. $\mathrm{ap}_{\mathrm{id}_A}(p) = p$.

Proof. All these properties follow from path induction on p (and q). □

Concerning the latter two items, it would be helpful to have the stronger statements that $\mathrm{ap}_{g \circ f} = \mathrm{ap}_g \circ \mathrm{ap}_f$, and likewise $\mathrm{ap}_{\mathrm{id}_A} = \mathrm{id}_{x=y}$. However, proving these statements would require to be able to conclude that two functions are equal as soon as they take on equal values on all points; this is something which we cannot prove yet!

A natural question now is, what happens if instead of $f : A \to B$ we consider a dependent function $f : \prod_{x:A} B(x)$? In this case, things are a bit more complicated: if we have $p : x =_A y$, then $f(x) : B(x)$ and $f(y) : B(y)$, so that it does not actually make sense to assert that $f(x) = f(y)$. So, before solving this puzzle, we need to understand dependent types a little better; we consider this below.

Types as higher groupoids

For paths $p, q : x = y$, we can also consider a path between these paths, $\alpha : p = q$. More rigorously, we should indicate the base type over which the identity type $p = q$ is formed, and then we have to write $\alpha : p =_{x =_A y} q$. We will also call such a path between paths a *2-path*.

Suppose that we have, in addition to this data, also another path $r : y = z$. Geometrically, the situation looks like this:

$$\begin{tikzcd} x \arrow[r, bend left, "p"{name=P}] \arrow[r, bend right, "q"'{name=Q}] \arrow[from=P, to=Q, Rightarrow, "\alpha"] & y \arrow[r, "r"] & z \end{tikzcd}$$

Using path induction on r, we can concatenate this data to a 2-path of type $p \cdot r = q \cdot r$ which we denote by $\alpha \blacklozenge r$,

$$\begin{tikzcd} x \arrow[r, bend left, "p \cdot r", ""{name=U, below}] \arrow[r, bend right, "q \cdot r"', ""{name=D}] & z \arrow[Rightarrow, from=U, to=D, "\alpha \blacklozenge r"] \end{tikzcd}$$

More precisely, there is a function which concatenates any 2-path to any 1-path in the sense of having the type

$$\_ \blacklozenge \_ : \prod_{A:\mathcal{U}} \prod_{x,y,z:A} \prod_{p,q:x=y} \prod_{\alpha:p=q} \prod_{r:y=z} (p \cdot r) = (q \cdot r).$$

Again, we only consider α and r as the arguments of this function, since the others can be deduced. Similarly, if we are given another $w : A$ and $o : w = x$ as in

$$\begin{tikzcd} w \arrow[r, "o"] & x \arrow[r, bend left, "p", ""{name=U, below}] \arrow[r, bend right, "q"', ""{name=D}] & y \arrow[Rightarrow, from=U, to=D, "\alpha"] \end{tikzcd}$$

then we can likewise use path induction on o in order to define a new 2-path $o \blacklozenge \alpha : o \cdot p = o \cdot q$.

What operations can we construct which concatenate two 2-paths to a new 2-path? For one, if we have $r, p, q : x = y$ and $\alpha : p = q$ and $\beta : q = r$ like this,

$$\begin{tikzcd} x \arrow[r, bend left=60, "p", ""{name=U, below}] \arrow[r, "q" description, ""{name=M}] \arrow[r, bend right=60, "r"', ""{name=D}] & y \arrow[Rightarrow, from=U, to=M, "\alpha"] \arrow[Rightarrow, from=M, to=D, "\beta"] \end{tikzcd}$$

then we already know that we can form the concatenation $\alpha \cdot \beta : p = r$. Since this is only one kind of concatenation of 2-paths, it is usually called the *vertical concatenation*, due to how the α and β are arranged in the diagram above. On the other hand, if we have this situation,

$$\begin{tikzcd} x \arrow[r, bend left, "p", ""{name=U1, below}] \arrow[r, bend right, "q"', ""{name=D1}] & y \arrow[r, bend left, "r", ""{name=U2, below}] \arrow[r, bend right, "s"', ""{name=D2}] & z \arrow[Rightarrow, from=U1, to=D1, "\alpha"] \arrow[Rightarrow, from=U2, to=D2, "\beta"] \end{tikzcd}$$

we can also construct a *horizontal concatenation* by defining

$$\alpha \star \beta :\equiv (\alpha \blacklozenge r) \cdot (q \blacklozenge \beta),$$

which is of type $(p \cdot r) = (q \cdot s)$. Alternatively, it would also have been possible to define this horizontal concatenation as $(p \blacklozenge \beta) \cdot (\alpha \blacklozenge s)$; the result would have been again equal, since one can find an element of the identity type

$$(\alpha \blacklozenge r) \cdot (q \blacklozenge \beta) = (p \blacklozenge \beta) \cdot (\alpha \blacklozenge s)$$

by using several consecutive path inductions.

Theorem 10. *(Exchange law) In a situation like this,*

$$\begin{tikzcd}
x \arrow[r, "p" description, ""{name=P}] \arrow[r, bend left=60, "o", ""{name=O, below}] \arrow[r, bend right=60, "q"', ""{name=Q, above}] & y \arrow[r, "s" description, ""{name=S}] \arrow[r, bend left=60, "r", ""{name=R, below}] \arrow[r, bend right=60, "t"', ""{name=T, above}] & z
\arrow[Rightarrow, from=O, to=P, "\alpha"]
\arrow[Rightarrow, from=P, to=Q, "\beta"]
\arrow[Rightarrow, from=R, to=S, "\gamma"]
\arrow[Rightarrow, from=S, to=T, "\delta"]
\end{tikzcd}$$

we have

$$(\alpha \cdot \beta) \star (\gamma \cdot \delta) = (\alpha \star \gamma) \cdot (\beta \star \delta).$$

Proof. As usual, we apply path induction, this time on α, β, γ and δ, which reduces the problem to this case:

$$\begin{tikzcd}
x \arrow[r, "p" description, ""{name=P}] \arrow[r, bend left=60, "p", ""{name=O, below}] \arrow[r, bend right=60, "p"', ""{name=Q, above}] & y \arrow[r, "s" description, ""{name=S}] \arrow[r, bend left=60, "s", ""{name=R, below}] \arrow[r, bend right=60, "s"', ""{name=T, above}] & z
\arrow[Rightarrow, from=O, to=P, "\mathrm{refl}_p"]
\arrow[Rightarrow, from=P, to=Q, "\mathrm{refl}_p"]
\arrow[Rightarrow, from=R, to=S, "\mathrm{refl}_s"]
\arrow[Rightarrow, from=S, to=T, "\mathrm{refl}_s"]
\end{tikzcd}$$

In this case, the desired equality is

$$(\mathrm{refl}_p \cdot \mathrm{refl}_p) \star (\mathrm{refl}_s \cdot \mathrm{refl}_s) = (\mathrm{refl}_p \star \mathrm{refl}_s) \cdot (\mathrm{refl}_p \star \mathrm{refl}_s).$$

Since we have $\mathrm{refl}_p \cdot \mathrm{refl}_p \equiv \mathrm{refl}_p$, and similar for refl_s and $(\mathrm{refl}_p \star \mathrm{refl}_s)$, this simplifies to

$$\mathrm{refl}_p \star \mathrm{refl}_s = \mathrm{refl}_p \star \mathrm{refl}_s,$$

which is true again by reflexivity. □

There are many more operations on higher-dimensional paths which can be constructed by path induction. We will stop our short discussion of this aspect by noting that all these higher operations can be regarded as one possible definition of an algebraic structure called an ∞-*groupoid*.

Dependent types as fibrations

We already learnt that we can think of a type A as a space with elements $x : A$ as "points" and equalities $p : x = y$ as "paths". We also just realized that a function $f : A \to B$ can be interpreted as a continuous map from the space A to the space B.

Now suppose that we apply this in the case $B \equiv \mathcal{U}$. So, the universe $\mathcal{U}$ is itself a gigantic space whose elements are themselves other spaces. (Such a "space of spaces" is often also called a *moduli space*.) So, a dependent type over A or type family over A is a function $D : A \to \mathcal{U}$; intuitively, this means that D assigns to every $x : A$ another space $D(x) : \mathcal{U}$ in a way which depends continuously on x. Conventionally, such a "dependent space" is known as a *fibration*; so, a type family is the HoTT analogue of the notion of fibration. As an example, if we start with the base space A being a circle, and attach to each $x : A$ a copy of the interval, such that as we go once around the circle, the interval comes back with a "twist", then we obtain the Möbius strip of Figure 1. What is depicted is not the function $D : A \to \mathcal{U}$, but rather the *total space* $\sum_{x:A} D(x)$, which is the union of

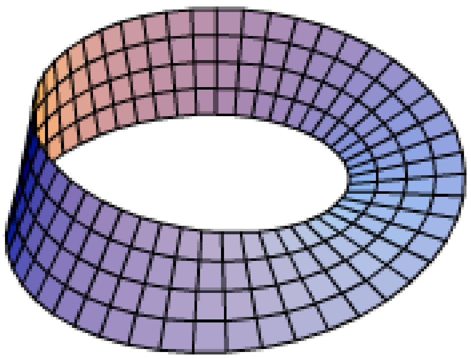

Figure 1: A Möbius strip

all the individual $D(x)$. One can identify the circle S^1 with the central line that winds around the strip; for every x, the space $D(x)$ then consists of the interval perpendicular to that circle. Higher inductive types will let us permit the definition of a type modelling the circle and a type modelling the interval, and then the Möbius strip will be an actual type family that we can construct. Until then, it is nothing but an illustration.

So, the dependent sum $\sum_{x:A} D(x)$ corresponds to the total space of the fibration $D : A \to \mathcal{U}$; but what about the dependent product $\prod_{x:A} D(x)$? Intuitively, an element $s : \prod_{x:A} D(x)$ corresponds to an assignment $x \mapsto s(x)$ with $s(x) : D(x)$ such that the dependence on x is continuous. Under the correspondence to fibrations, such an s is conventionally known as a *section*. The Möbius strip is very special in that it permits a section which embeds the circle as the "base" into the total space $\sum_{x:A} D(x)$; in general, this is not the case.

With this intuition in mind, we can now move on to discussing the behavior of paths with respect to such fibrations.

Theorem 11. *Given a type family $D : A \to \mathcal{U}$ over A and a path $p : x =_A y$, there exists a function*

$$\mathrm{transport}^D(p) \,:\, D(x) \to D(y).$$

Proof. Path induction on p, using the definition $\mathrm{transport}^D(\mathrm{refl}_x) \equiv \mathrm{id}_{D(x)}$. □

Whenever the type family D is clear, then we also use the shorthand notation $p_* : D(x) \to D(y)$ in place of $\mathrm{transport}^D(p)$.

If $D : A \to \mathcal{U}$ is interpreted as a proposition depending on $x : A$, then we can say that $x : A$ "has property D" if $D(x)$ is inhabited. Therefore, the theorem tells us that if x is equal to y, and x has property D, then so does y. More precisely, any witness or proof of the statement "x has property D" can be transported into a witness or proof of the statement "y has property D".

In the topological theory of fibrations, the transport function can be interpreted as a connection, which is a fundamental structure in *differential* geometry. In this sense, a type family is more than just a fibration: it is a fibration equipped with a connection.

Important properties of the transport function are the following, which are all easily proven by the usual path induction:

Theorem 12. *If $q : y = z$ is another path, then*

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

for all $t : D(x)$.

So the transport map has the expected behaviour upon concatenation of paths; note the similarity to the previous theorem $\mathrm{ap}_f(q \cdot p) = \mathrm{ap}_f(q) \cdot \mathrm{ap}_f(p)$.

Using transport, we can not only transport elements around, we can also transport paths themselves,

$$\mathrm{ap}_{\mathrm{transport}^D} : \prod_{s,t:D(x)} (s = t) \to (p_*(s) = p_*(t)). \tag{3}$$

The behaviour of transport with respect to changing the "base" type A is like this:

Theorem 13. *If $f : B \to A$ is a function, then $D \circ f : B \to \mathcal{U}$ is again a type family, and*

$$\mathrm{transport}^{D \circ f}(p, t) = \mathrm{transport}^D(\mathrm{ap}_f(p), t).$$

for all $p : x =_B y$ with $x, y : B$, and $t : D(f(x))$.

Finally, we also note the behaviour of transport with respect to changing the type family, while keeping the base A fixed:

Theorem 14. *Given any two type families $D, E : A \to \mathcal{U}$ and a family of dependent functions $f : \prod_{x:A} D(x) \to E(x)$ and any $p : x =_A y$ and $t : D(x)$, we have*

$$\mathrm{transport}^E(p, f(x, t)) = f(y, \mathrm{transport}^D(p, t)).$$

We can express this as saying that transport "commutes" with the application of a dependent function. In terms of category theory, f is a transformation between functors, and the theorem shows that this transformation is *automatically* natural. This is similar to how Lemma 3 shows that every function is automatically continuous.

If we apply transport in the case of a non-dependent type, it acts trivially:

Theorem 15. *If $D \equiv \lambda x.B : A \to \mathcal{U}$ for some fixed $B : \mathcal{U}$, then for any $x, y : A$ and $p : x =_A y$ there is a family of paths*

$$\mathrm{transportconst}^B(p) : \prod_{t:B} \mathrm{transport}^{\lambda x.B}(p, t) = t$$

We are now in a situation to generalize the function $\mathrm{ap} : (x = y) \to (f(x) = f(y))$ to the case in which f is a dependent function, by simply transporting $f(x)$ from $D(x)$ to $D(y)$ before comparing it to $f(y)$:

Theorem 16. *Given any dependent function $f : \prod_{x:A} D(x)$, there exists a function*

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

Again we consider the elements $x, y : A$ to be implicit arguments of apd_f for which we omit mention. The proof of the theorem is again straightforward by path induction on p. While the previous ap was a function $(x = y) \to (f(x) = f(y))$, the new apd_f for dependent f is itself a dependent function. Also apd_f has the expected behaviour with respect to concatenation of paths, and more generally there are more complicated analogues of Theorem 9, but we omit these.

When f is a non-dependent function, we would expect apd_f to reduce to ap_f, assuming that we identify $f : A \to B$ with the corresponding element of $\prod_{x:A} B$. This is indeed the case:

Theorem 17. *For all $f : A \to B$ and $p : x =_A y$, we have*

$$\mathrm{apd}_f(p) = \mathrm{transportconst}^B(p, f(x)) \cdot \mathrm{ap}_f(p).$$

The is well-typed since the path on the left goes from $\mathrm{transport}^{\lambda x.B}(p, f(x))$ to $f(y)$, while the one on the right is a concatenation of a path from $\mathrm{transport}^{\lambda x.B}(p, f(x))$ to $f(x)$ and a path from $f(x)$ to $f(y)$. As usual, the proof is by path induction on p, in which case both sides reduce to $\mathrm{refl}_{f(x)}$.

Path lifting

The previous considerations involving transport were about turning a path $p : x = y$ on the "base" A into a function $\mathrm{transport}^D(p) : D(x) \to D(y)$. But can we also lift a path in the base to a path in D? In order to make sense of this, we need to consider paths in the total space $\sum_{x:A} D(x)$, and then we have:

Theorem 18 (Path lifting property)**.** *For any $x : A$ and $t : D(x)$, every path $p : x =_A y$ gives a path*

$$\mathrm{lift}(t, p) \,:\, (x, t) =_{\sum_{x:A} D(x)} (y, p_*(t)),$$

and this path lies over p in the sense that

$$\mathrm{ap}_{\mathrm{pr}_1}(\mathrm{lift}(t, p)) = p.$$

Proof. Using path induction on p reduces the proof of both statements to the case $y \equiv x$ and $p \equiv \mathrm{refl}_x$, for which they are trivial. ☐

Omnipotent path induction?

The previous results may suggest that path induction is a proof method with which one can prove almost anything, since it always reduces the proof to a trivial case. However, this is by no means the case: for example, the statement

$$\prod_{x:A} \prod_{p:x=x} p = \mathrm{refl}_x \tag{4}$$

is *not* derivable by path induction. In fact, path induction does not apply here since it would require at least one endpoint of p to be free—based path induction applies to statements of the form $\prod_{x:A} \prod_{p:x=y} C(x, y, p)$ only! We will see soon that together with the univalence axiom, one can show (4) to be false. Actually, (4) is closely related to Axiom K, which is a commonly imposed axiom resulting in so-called *extensional* type theory, in which two things can be equal in at most one way. Due to the incompatibility with the univalence axiom, and the realization that univalence is a more powerful principle resulting in a foundation of mathematics close to mathematical practice, we do not impose Axiom K.

Example: discriminating the elements of `Bool`

In order to illustrate the use of the results derived here, we show that the two canonical elements of `Bool` are not equal:

Theorem 19. $(0 = 1) \to \mathbf{0}$.

Proof. Define the type family $D : \texttt{Bool} \to \mathcal{U}$ by induction over `Bool` using the values

$$D(0) :\equiv \mathbf{0}, \qquad D(1) :\equiv \mathbf{1}$$

on the base cases. We know that $D(1)$ is inhabited by $* : D(1)$. Hence for any putative equation $p : 0 = 1$, we have

$$(p^{-1})_*(*) \,:\, D(0),$$

which is an element of $\mathbf{0}$ since $D(0) \equiv \mathbf{0}$. □