15-819 Homotopy Type Theory Lecture Notes

Robert Lewis and Joseph Tassarotti

October 21 and 23, 2013

1 Paths-over-Paths

Recall that last time we explored the higher groupoid structure of types, and showed that for non-dependent maps, ap preserves this structure. Now, in the case where we have a dependent function $f : \Pi x : A.B$, we would like to similarly state that f maps equals to equals, so that given a path $p : \mathsf{Id}_A(M, N)$, there is some map which takes in p and gives a path between fM and fN. However, because f is dependent, $fM : [M/x]B$ and $fN : [N/x]B$. Although these types are related, they are not equal, so we cannot talk about propositional equality between fM and fN.

In earlier lectures, we defined $\mathsf{tr}[x.B]p : [M/x]B \to [N/x]B$, often written as p_*, which lifts the path p to a mapping between the fibers $[M/x]B$ and $[N/x]B$. Since $p_*(fM)$ and fN share the same type, we can meaningfully talk about equality between them. We can now define a map $\mathsf{apd}_f : \Pi p : \mathsf{Id}_A(M, N).\mathsf{Id}_{[N/x]B}(p_*(fM), fN)$ by

$$\mathsf{apd}_f p := \mathsf{J}[m.n.z.\mathsf{Id}_{[n/x]B}(z_*(fm), fn)](p; m.\mathsf{refl}_{[m/x]B}(m))$$

This has the appropriate type because when the path is simply $\mathsf{refl}_A(M)$, we have that $(\mathsf{refl}_A(M))_* \equiv \mathsf{refl}_{[M/x]B}(fM)$. See figure 1 for a pictorial representation of this.

Now, since p_*^{-1} gives a map between the fibers going the other way, we could just as well have defined an analogous term $\mathsf{apd}'_f : \mathsf{Id}_A(M, N) \to \mathsf{Id}_{[M/x]B}(fM, p_*^{-1}(fN))$. Moreover, we have that

$$\begin{aligned} \mathsf{ap}_{p_*^{-1}}(\mathsf{apd}_f p) \quad &: \quad \mathsf{Id}_{[M/x]B}(p_*^{-1}(p_*(fM)), p_*^{-1}(fN)) \\ &\equiv \mathsf{Id}_{[M/x]B}(fM, p_*^{-1}(fN)) \end{aligned}$$

$$\begin{aligned} \mathsf{ap}_{p_*}(\mathsf{apd}_f p^{-1}) \quad &: \quad \mathsf{Id}_{[N/x]B}(p_*(fM)), p_*(p_*^{-1}(fN))) \\ &\equiv \mathsf{Id}_{[N/x]B}(p_*(fM), fN) \end{aligned}$$

which shows that these two theorems imply one another.

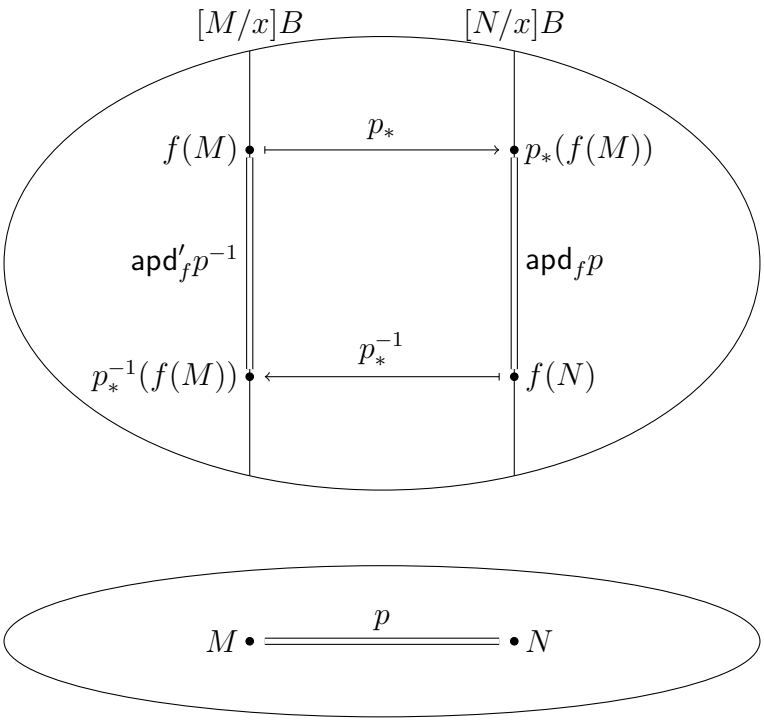

Figure 1: Paths-over-paths

The lack of symmetry in the types of apd_f and apd'_f is somewhat awkward when developing machine checked proofs. It's more convenient to define a symmetric notation, $f(M) =_p^{x.B} f(N) :\equiv Id[N/x]B(p_*(fM), fN)$, which we read as "$f(M)$ and $f(N)$ are correlated by p" . This corresponds to the type of paths *over* the path p. Using this notation, we can prove theorems about this type like:

$$\mathrm{sym}_{\mathrm{corr}} \quad : \quad Q =_p^{x.B} R \;\to\; R =_{p^{-1}}^{x.B} Q$$

$$\mathrm{trans}_{\mathrm{corr}} \quad : \quad Q =_p^{x.B} R \;\to R =_q^{x.B} S \to Q =_{p \cdot q}^{x.B} S$$

2 Equivalence of Types

2.1 Motivation

We start by informally recalling some notions of equivalence that are commonly used in mathematics:

1. *Biconditional propositions*: Given two propositions p and q such that $p \supset q$ and $q \supset p$, we might wish to say that $p = q$, because these two propositions are logically equivalent. In classical logic, this makes sense, because p and q are both either equal to true or equal false. To quote Whitehead and Russell [3, p.115]:

   > When each of two propositions implies the other, we say that the two are *equivalent*, which we write "$p \equiv q$" ... It is obvious that two propositions are equivalent when, and only when, both are true or both are false...
   >
   > We shall give the name of a *truth-function* to a function $f(p)$ whose argument is a proposition, and whose truth-value depends only upon the truth-value of its argument. All the functions of a proposition with which we shall be specially concerned will be truth-functions, *i.e.* we shall have
   >
   > $$p \equiv q \,.\supset\, .f(p) \equiv f(q).$$

   This means that for Whitehead and Russell, if p and q are logically equivalent, then they are indiscernible. However, in the proof relevant setting of type theory, this is not the case, because these types classify particular pieces of data. Although terms of the type $f : p \to q$ and $g : q \to p$ give us ways to interconvert proofs of p and q, a proof of p is not by itself a proof of q. Moreover, it need not even be the case that f and g are inverses of each other.
2. *Isomorphic sets*: In set theory, we say that two sets A and B are *isomorphic* if there is a bijection between them. That is, there are functions $f : A \to B$ and $g : B \to A$ such that $g(f(a)) = a$ and $f(g(b)) = b$. In many contexts, it is not relevant for us to distinguish between isomorphic sets. However, in ZF set theory, just because two sets are isomorphic does not mean they are indiscernible, so we cannot regard them as equal.
   This is a larger symptom of the fact that although ZF set theory lets us encode the structures of mathematics, it does not support abstraction. Propositions like $0 \in 1$ are perfectly well-formed, and are even true for most encodings of the natural numbers as sets. As de Bruijn points out [1], these artifacts of a particular

encoding contradict the way we conceptually think of mathematics[1]:

> In our mathematical culture we have learned to keep things apart. If we have a rational number and a set of points in the Euclidean plane, we cannot even imagine what it means to form the intersection. The idea that both might have been coded in ZF with a coding so crazy that the intersection is *not empty* seems to be ridiculous...
>
> A very clear case of thinking in terms of types can be found in Hilbert's axiomatization of geometry. He started by saying that he assumes there are certain things which will be called *points* and certain things to be called *lines*. Nothing is said about the nature of these things.

Type theory rules out statements like $0 \in 1$ as ill-formed. As we shall see, this same facility for abstraction allows us to give a more suitable treatment of equivalence.

Now, we turn to the question of equivalences of types. Applying our naïve intuition of regarding types as sets, we might say that types are isomorphic precisely when there is a bijection between them. In ITT, this will work for types corresponding to first order data, but we encounter problems when considering functions.

More precisely, to show that $A \to B$ is isomorphic to $C \to D$, we need to construct functions $F : (A \to B) \to (C \to D)$ and $G : (C \to D) \to (A \to B)$ such that for all $f : A \to B$ and $g : C \to D$, $G(F(f)) = f$ and $F(G(g)) = g$. In a set-theoretic setting, it would suffice to show that for all $x \in A$, $G(F(f))(x) = f(x)$, and similarly for $F \circ G$. However, in ITT we lack function extensionality, this is not enough. We need to show that $G \circ F$ maps f precisely back to itself. One might try to resolve this by quotienting by extensionality or adding in an axiom of extensionality.

However, the problem becomes even more difficult when considering universes. We would need to show that for each type A in the universe, $G(F(A)) = A$. Just as with functions, where we were really interested in showing that $G \circ F$ mapped a function to something that was extensionally equivalent, here we want $G(F(A))$ to itself be isomorphic to A not equal.

2.2 Homotopy Equivalence

We now introduce the notion the notion of a *homotopy*. Given two functions $f, g : A \to B$, a homotopy from f to g is a term with type $\Pi x : A.\mathsf{Id}_B(fx, gx)$. We

[1] A portion of this passage is quoted in [2], which contains an interesting discussion about some advantages and disadvantages of types and sets.

introduce the notation $f \sim_{A\to B} g$ for the type of homotopies from f to g. If this type is inhabited, we say that f is "homotopic to" g.

Now, given $H : f \sim_{A\to B}$, we have that for all $x : A$, $fx =_B gx$. But in fact there is something more going on: H is *dependently functorial* in $x : A$. That is, H respects paths between inhabitants in A and B. This property is also called *naturality*; we can say that H is a sort of polymorphism in $x : A$. This means that the following diagram commutes:

$$\begin{array}{ccc} f(a) & \overset{H(a)}{=\!=} & g(a) \\ {\scriptstyle \mathsf{ap}_f(p)}\Big\| & & \Big\|{\scriptstyle \mathsf{ap}_g(p)} \\ f(a') & \underset{H(a')}{=\!=} & g(a') \end{array}$$

2.3 Basic Properties of Equivalence

For a function $f : A \to B$, we define an *equivalence* between A and B, by

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

The proposition expressing that two types A and B are equivalent, written $A \simeq B$ b, is

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

Since we are in a proof-relevant setting, the information that $A \simeq B$ consists of five things:

- A function $f : A \to B$
- A function $g : B \to A$
- A proof $\alpha : \Pi y : B.f(g(y)) =_B y$
- A function $h : B \to A$
- A proof $\beta : \Pi x : A.h(f(x)) =_A x$

To prove that $A \simeq B$, we need to provide all of these as evidence, and from evidence that $A \simeq B$, we can extract all of these.

We will also be interested in the notion of a *quasi-inverse*:

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

As one might hope, the notions of equivalence and quasi-inverse are very closely related. One can prove the following properties:

1. For every $f : A \to B$, there is a function $\mathsf{qinv}(f) \to \mathsf{isequiv}(f)$.
2. For every $f : A \to B$, there is a function $\mathsf{isequiv}(f) \to \mathsf{qinv}(f)$.

This means that the two notions are logically equivalent: a function is an equivalence if and only if it has a quasi-inverse. In addition, we can show that $\mathsf{isequiv}(f)$ expresses an HPROP: that is, up to higher homotopy, there is only one proof of this fact. This will become important later.

2.4 Function extensionality

The axiom of function extensionality allows us to show that the $(f =_{A\to B} g) \simeq (f \sim_{A\to b} g)$. Even without the axiom, we can define the map

$$\mathsf{happly} : f =_{A\to B} g \to f \sim_{A\to B} g$$

Now, the axiom says that the above map is an equivalence: if we have a proof of $f \sim_{A\to B} g$, we may assume that we have a proof of $f =_{A\to B} g$. This is not necessarily provable without the axiom. For example, in the natural number type, $\lambda x.0 + x \sim_{N\to N} \lambda x.x$, but since addition was defined inductively on the second argument, we cannot find a path between them.

2.5 Exercises

The following propositions are left as exercises, with the first one begun for explanatory purposes:

1. Show that $\mathsf{id}_A : A \to A$ is an equivalence.
   To do this, we need four pieces of information:
   (a) $g : A \to A$. Take this to be id_A.
   (b) A proof $\alpha : \Pi y : A.\mathsf{id}_A(g(y)) =_A y$.
   (c) $h : A \to A$. Again, take this to be id_A.
   (d) A proof $\beta : \Pi x : A.h(\mathsf{id}_A(x)) =_A x$.
2. If $f : A \to B$ is an equivalence, then there is $f^{-1} : B \to A$ (given by the quasi-inverse of f) that is also an equivalence.
3. If $f : A \to B$ and $g : B \to C$ are equivalences, then so is $g \circ f : A \to C$.

3 Structure of Paths in Types

We want to examine the paths inside certain types. For the negative types, this will be relatively simple. For the positive types, it will be much harder. There are many outstanding open problems within the positive types.

3.1 Product Types

We start by examining the paths in $\mathsf{Id}_{A\times B}(\_,\_)$.

There is a function f such that

$$f : \mathsf{Id}_{A\times B}(x,y) \to (\mathsf{Id}_A(\pi_1 x, \pi_1 y) \times \mathsf{Id}_B(\pi_2 x, \pi_2 y)).$$

Specifically,

$$f :\equiv \lambda p.\langle \mathsf{ap}_{\pi_1}(p), \mathsf{ap}_{\pi_2}(p)\rangle$$

Roughly speaking, if $x =_{A\times B} y$, then $\pi_1 x =_A \pi_1 y$ and $\pi_2 x =_B \pi_2 y$.

Proposition. *f is an equivalence:* $\mathsf{Id}_{A\times B}(x,y) \simeq \mathsf{Id}_A(\pi_1 x, \pi_1 y) \times \mathsf{Id}_B(\pi_2 x, \pi_2 y)$.

Proof. As noted in Section 2, it suffices to produce a quasi-inverse for f. We need to construct three objects:

1. $g : (\mathsf{Id}_A(\pi_1 x, \pi_1 y) \times \mathsf{Id}_B(\pi_2 x, \pi_2 y)) \to \mathsf{Id}_{A\times B}(x,y)$
2. $\alpha : g(f(p)) =_{\mathsf{Id}_{A\times B}(x,y)} p$
3. $\beta : f(g(q)) =_{\mathsf{Id}_A(\pi_1 x,\pi_1 y)\times \mathsf{Id}_B(\pi_2 x,\pi_2 y)} q$

We construct these as follows:

1. We define two auxiliary functions

$$\mathsf{pair} :\equiv \lambda x \lambda y \langle x, y\rangle : A \to B \to A \times B$$

and

$$\mathsf{ap2}_f : \mathsf{Id}(x,x') \to \mathsf{Id}(y,y') \to \mathsf{Id}(fxy, fx'y')$$

Using these, we can then define

$$g :\equiv \lambda\langle p, q\rangle.\mathsf{ap2}_{\mathsf{pair}} pq$$

.

2. To define α, it suffices (by FUNEXT) to show:
   - $\eta : \Pi p(\mathsf{ap2}_{\mathsf{pair}}(\mathsf{ap}_{\pi_1}(p), \mathsf{ap}_{\pi_2}(p)) = p)$
   - $\beta_1 : \Pi p \Pi q(\mathsf{ap}_{\pi_1}(\mathsf{ap2}_{\mathsf{pair}} pq) = p)$
   - $\beta_2 : \Pi p \Pi q(\mathsf{ap}_{\pi_2}(\mathsf{ap2}_{\mathsf{pair}} pq) = q)$

   By path induction, we need to find R such that

$$x : A \times B \vdash R : (\mathsf{ap2}_{\mathsf{pair}}(\mathsf{ap}_{\pi_1}(\mathsf{refl}(x)), \mathsf{ap}_{\pi_2}(\mathsf{refl}(x)))) = \mathsf{refl}(x)$$

   Then,

$$\eta :\equiv J[\,](p; x.R).$$

By our earlier definition of ap^2, we have that

$$\begin{aligned}
\mathsf{ap}_{\pi_1}(\mathsf{refl}(x)) &\equiv \mathsf{refl}(\pi_1(x)) \\
\mathsf{ap}_{\pi_2}(\mathsf{refl}(x)) &\equiv \mathsf{refl}(\pi_2(x)), \text{ and from these,} \\
\mathsf{ap2}_{\mathsf{pair}}(\mathsf{refl}(\pi_1(x)))(\mathsf{refl}(\pi_2(x))) &\equiv \mathsf{refl}\langle \pi_1(x), \pi_2(x)\rangle \\
&\equiv \mathsf{refl}(x)
\end{aligned}$$

3. The constructions of β_1 and β_2 are similar and left as exercises.

□

3.2 Coproduct Types

Similarly, we can look into $\mathsf{Id}_{A+B}(x, y)$. Intuitively speaking, we would like to say that any path in the space $A + B$ is either a path in A or a path in B; we would never expect to have a path (equation) between an $\mathsf{inl}(a)$ and an $\mathsf{inl}(b)$.

We would like to prove the following facts:

$$\begin{aligned}
\mathsf{Id}_{A+B}(\mathsf{inl}(a), \mathsf{inl}(a')) &\simeq \mathsf{Id}_A(a, a') \\
\mathsf{Id}_{A+B}(\mathsf{inr}(b), \mathsf{inr}(b')) &\simeq \mathsf{Id}_B(b, b') \\
\mathsf{Id}_{A+B}(\mathsf{inl}(a), \mathsf{inr}(b)) &\simeq 0 \\
\mathsf{Id}_{A+B}(\mathsf{inr}(a), \mathsf{inl}(b)) &\simeq 0
\end{aligned}$$

Proving this requires a bit of a trick.

Suppose we wanted to prove the first equivalence alone. The right-to-left direction is simple. For the left-to-right direction, we need to exhibit

$$p : \mathsf{Id}_{A+B}(\mathsf{inl}(a), inl(a')) \vdash R : \mathsf{Id}_A(a, a').$$

R must be a path induction on p, of the form $R = J[C](p, \_)$ for some motive C. The conclusion of this path induction will be of the form $C(\mathsf{inl}(a), \mathsf{inl}(a'), p)$. But what we need is $\mathsf{Id}_A(a, a')$ (note the lack of inl). One might try to define something like $D(u, v) = \mathsf{Id}_A(\mathsf{outl}(u), \mathsf{outl}(v))$, but this cannot exist, since outl cannot be a total function.

[2] This is a striking example of anti-modularity. One has no reason to expect that this equality should hold definitionally; it depends essentially on how ap was defined, not just on its type. It would be nice to avoid this kind of code-on-code dependency, since "the proof should not have to know about the computation."

This approach, then, will not work. Instead, we must take a different approach. We will find a motive $F : (A + B) \times (A + B) \to \mathcal{U}$ such that:

$$\begin{aligned} F(\mathsf{inl}(a), \mathsf{inl}(a')) &\equiv \mathsf{Id}_A(a, a') \\ F(\mathsf{inr}(a), \mathsf{inr}(a')) &\equiv \mathsf{Id}_B(b, b') \\ F(\mathsf{inl}(a), \mathsf{inr}(b)) &\equiv 0 \\ F(\mathsf{inr}(a), \mathsf{inl}(b)) &\equiv 0 \end{aligned}$$

Such an F expresses all of the desired properties of the coproduct.

Exercise. *Define such an F by "double induction."*

The following lemma expresses the subgoal of our path induction with motive F:

Lemma. $x : A + B \vdash \_ : F(x, x)$.

Proof. Our proof of this will be a case statement:

$$\underline{\mathsf{case}}[z.F(z, z)](x; m : A.\mathsf{refl}_A(m), n : B.\mathsf{refl}_B(n)) : F(x, x)$$

Note that $\mathsf{refl}_A(m) : [\mathsf{inl}(m)/z]F(z, z)$, since

$$\mathsf{Id}_{A+B}(m, n) \equiv F(\mathsf{inl}(m), \mathsf{inl}(m)) \equiv [\mathsf{inl}(m)/z]F(z, z).$$

□

To complete the proof, we must define something of the type

$$\Pi x : A + B \Pi x' : A + B(\mathsf{Id}_{A+B}(x, y) \to F(x, x')).$$

This is our task for next time!

References

[1] N. G. de Bruijn. On the roles of types in mathematics. In Philippe de Groote, editor, *The Curry-Howard Isomorphism*, volume 8 of *Cahiers du Centre de Logique*. Academia, 1995.

[2] Leslie Lamport and Lawrence C. Paulson. Should your specification language be typed. *ACM Trans. Program. Lang. Syst.*, 21(3):502–526, 1999.

[3] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*, volume 1. Cambridge University Press, 1963. Reprint of the 1927 second edition. Accessed at `https://archive.org/details/PrincipiaMathematicaVolumeI`.