Topic 6: Identity types

May 30, 2014

We can now move on to one of the central concepts which turns type theory into *homotopy* type theory, namely the concept of *identity types*: for every type A and $x, y : A$, there is an identity type $x =_A y$ which turns the statement that x is equal to y into a type itself, as one would expect from the propositions-as-types correspondence. The elements of this type can be thought of as witnesses of the equality of x and y. One of the main features of identity types, which it has in common with homotopy, is that one can iterate them: for two witnesses $p, q : x =_A y$, one can ask whether these two witnesses are themselves equal by considering the identity type $p =_{x =_A y} q$.

Strictly speaking, there are two different definitions of identity types. Both of these are indexed by $A : \mathcal{U}$ and by the two elements $x, y : A$ that one wants to compare. In other words, both of these variants have type formers given by functions $\prod_{A:\mathcal{U}} A \to A \to \mathcal{U}$. In order to distinguish these two inductive type families, we start by writing these identity types as $\mathrm{id}_u(A, x, y)$ and $\mathrm{id}_b(A, x, y)$, where the subscript disambiguates the two different versions, called 'based' and 'unbased' identity types.

Based identity types. The based identity type $\mathrm{id}_b : \prod_{A:\mathcal{U}} A \to A \to \mathcal{U}$ has the first two arguments $A : \mathcal{U}$ and $x : A$ as parameters, while the third argument $y : A$ is an index.

More concretely, for every fixed $A : \mathcal{U}$ and $x : A$, the dependent type $\mathrm{id}_b(A, x) : A \to \mathcal{U}$ is an inductive type family with only one constructor,

- $\mathrm{refl}_x : \mathrm{id}_b(A, x, x)$,

which is a canonical witness of the equality of x to itself. Since there are no other constructors at all, this may seem very similar to the unit type—but as an inductive type *family*, it behaves quite differently and we will see that it equips the theory with a very rich structure.

The elimination rule then states the following: for every dependent type $C : \prod_{y:A} \mathrm{id}_b(A, x, y) \to \mathcal{U}$, one can construct an element of $C(y, p)$ for any given $y : A$ and $p : \mathrm{id}_b(A, x, y)$ by reducing to the 'base case' given by $y \equiv x$ and $p \equiv \mathrm{refl}_x$. In other words, there is an induction principle whose functional form looks like this:

$$\mathrm{ind}_{\mathrm{id}_b(A,x)} : \prod_{C : \prod_{y:A} \mathrm{id}_b(A,x,y) \to \mathcal{U}} C(x, \mathrm{refl}_x) \to \prod_{y:A} \prod_{p : \mathrm{id}_b(A,x,y)} C(y, p). \tag{1}$$

As an example application, we show that propositional equality is symmetric, meaning that we construct a dependent function

$$\prod_{x,y:A} \mathrm{id}_b(A, x, y) \to \mathrm{id}_b(A, y, x)$$

as follows. For given $x, y : A$ and $p : \mathrm{id}_b(A, x, y)$, we can apply the elimination rule in order to reduce to the case that $y \equiv x$ and $p \equiv \mathrm{refl}_x$, in which case it remains to construct an element of $\mathrm{id}_b(A, x, x)$, for which we can take again refl_x. More precisely, we have

$$\lambda x.\mathrm{ind}_{\mathrm{id}_b(A,x)}(\lambda y.\lambda p.\mathrm{id}_b(A, y, x), \mathrm{refl}_x) \; : \; \prod_{x,y:A} \mathrm{id}_b(A, x, y) \to \mathrm{id}_b(A, y, x).$$

So what does the induction principle (1) do, intuitively? In case in which we apply it to a dependent type $C : A \to \mathcal{U}$, i.e. a type whose dependence on $\mathrm{id}_b(A, x, y)$ is trivial, it specializes to an induction principle of the form

$$C(x) \to \prod_{y:A} \prod_{p:\mathrm{id}_b(A,x,y)} C(y),$$

which has a very simple interpretation: if we have an element of $C(x)$, then we can also obtain an element of $C(y)$ as soon as an identity between x and y is given; usually, applying such a procedure is known as *substitution*! Or, in logical terms: whenever $C(x)$ is a proposition, then the variable x can be *substituted* by any y equal to it, and this induction principle turns any proof of $C(x)$ into a proof of $C(y)$.

Finally, the computation rule says that if we apply the elimination rule in a 'base case', then we recover the given data. In other words,

$$\mathrm{ind}_{\mathrm{id}_b(A,x)}(C, d, x, \mathrm{refl}_x) \equiv d.$$

where $d : C(x, \mathrm{refl}_x)$ is arbitrary.

Unbased identity types. The other incarnation of identity types is the *unbased* identity type $\mathrm{id}_u : \prod_{A:\mathcal{U}} A \to A \to \mathcal{U}$, in which only the first argument is a parameter, while both other arguments are indices. More concretely, $\mathrm{id}_u(A) : A \to A \to \mathcal{U}$ is an inductive type family with again only one constructor,

- $\mathrm{refl} : \prod_{x:A} \mathrm{id}_u(A, x, x)$.

For consistency of notation with based identity types, we also write refl_x instead of $\mathrm{refl}(x)$, although this creates the ambiguity that the expression refl_x stands for the constructor both in the based and unbased cases. But since we will soon stop distinguishing between based and unbased identity types anyway, this ambiguity is actually desired. Hence, the functional form of the elimination rule takes the form

$$\mathrm{ind}_{\mathrm{id}_u(A)} : \prod_{C:\prod_{x,y:A} \mathrm{id}_u(A,x,y) \to \mathcal{U}} \left(\prod_{x:A} C(x, x, \mathrm{refl}_x) \right) \longrightarrow \prod_{x,y:A} \prod_{p:\mathrm{id}_u(A,x,y)} C(x, y, p).$$

The computation rule is this:

$$\mathrm{ind}_{\mathrm{id}_u(A)}(C, s, x, x, \mathrm{refl}_x) \equiv s,$$

where $s : \prod_{x:A} C(x, x, \mathrm{refl}_x)$ is arbitrary.

Equivalence of based and unbased path induction

So how do the based and unbased identity types relate? We will now show that the based identity type also satisfies the inference rules of the unbased identity type and vice versa, where both types are considered as inductive type families $\prod_{A:\mathcal{U}} A \to A \to \mathcal{U}$. This makes the situation analogous to the one for `Bool` vs. $\mathbf{1}+\mathbf{1}$, which also satisfy the same inference rules. Using the notion of equivalence and the upcoming univalence axiom, it will become possible to show that the types $\mathrm{id}_b(A,x,y)$ and $\mathrm{id}_u(A,x,y)$ are themselves equal, and so there is no need to distinguish between them.

We start by showing that the based identity type satisfies the inference rules of the unbased one. For the formation rule, this is trivial, since the formation rules are the same:

$$\mathrm{id}_b,\ \mathrm{id}_u : \prod_{A:\mathcal{U}} A \to A \to \mathcal{U}.$$

Likewise, when considering all three arguments as parameters/indices of the type family, even the introduction rules are the same:

$$\mathrm{refl} : \prod_{A:\mathcal{U}} \prod_{x:A} \mathrm{id}(A,x,x),$$

where id stands for either id_b or id_u.

For the introduction and computation rules, the situation is much less straighforward. Starting with the elimination rule (1), which we rewrite slightly differently by including the dependence on x explicitly,

$$\mathrm{ind}_{\mathrm{id}_b(A)} : \prod_{x:A} \prod_{C:\prod_{y:A} \mathrm{id}_b(A,x,y)\to\mathcal{U}} C(x,\mathrm{refl}_x) \to \prod_{y:A} \prod_{p:\mathrm{id}_b(A,x,y)} C(y,p), \tag{2}$$

we will derive an element of the corresponding unbased induction principle,

$$\prod_{D:\prod_{x,y:A} \mathrm{id}_b(A,x,y)\to\mathcal{U}} \left(\prod_{x:A} D(x,x,\mathrm{refl}_x)\right) \to \prod_{x,y:A} \prod_{p:\mathrm{id}_b(A,x,y)} D(x,y,p). \tag{3}$$

But this is easy to do, since given any $D : \prod_{x,y:A} \mathrm{id}_b(A,x,y) \to \mathcal{U}$, any $s : \prod_{x:A} D(x,x,\mathrm{refl}_x)$, and any $x : A$, we can construct an element of $D(x,y,p)$ by applying $\mathrm{ind}_{\mathrm{id}_b(A)}$ to x and the dependent type $C :\equiv D(x) : \prod_{y:A} \mathrm{id}_b(A,x,y) \to \mathcal{U}$, and thereby obtaining a function

$$\mathrm{ind}_{\mathrm{id}_b(A)}(x, D(x)) : D(x,x,\mathrm{refl}_x) \to \prod_{y:A} \prod_{p:\mathrm{id}_b(A,x,y)} D(x,y,p),$$

which gives us an element of $D(x,y,p)$ upon evaluating this function on $s(x)$, as desired. In conclusion, this means that the expression

$$\lambda D.\lambda s.\lambda x.\lambda y.\lambda p.\mathrm{ind}_{\mathrm{id}_b(A)}(x, D(x), s(x), y, p)$$

is an element of (3). It is now straightforward to show that this expression satisfies the appropriate computation rule: upon evaluating it on some D, s and x, and then on $y \equiv x$ and $p \equiv \mathrm{refl}_x$, one indeed recovers $s(x)$ by the computation rule for $\mathrm{ind}_{\mathrm{id}_b(A)}$.

Showing that $\mathrm{id}_u(A)$ also satisfies the based path induction is more difficult, since the first point $x : A$ in based path induction is fixed, so it is not clear how to apply unbased path induction, which can be applied only in cases in which we have an assumption of the form $\prod_{x:A} D(x,x,\mathrm{refl}_x)$. In order to get around this obstacle, it is helpful to consider *all instances* of based path induction at once, in the sense of considering the type

$$\prod_{x,y:A} \prod_{p:\mathrm{id}_u(A,x,y)} \prod_{C:\prod_{z:A} \mathrm{id}_u(A,x,z)\to\mathcal{U}} C(x,\mathrm{refl}_x) \to C(y,p), \tag{4}$$

which lives in the next higher universe $\mathcal{U}'$. By (unbased!) path induction, in order to construct an element of this type it is enough to do so in the case that $y \equiv x$ and $p \equiv \mathrm{refl}_x$, in which case we can simply take

$$\lambda x.\lambda C.\lambda c.c : \prod_{x:A} \prod_{C:\prod_{z:A} \mathrm{id}_u(A,x,z)\to\mathcal{U}} C(x,\mathrm{refl}_x) \to C(x,\mathrm{refl}_x)$$

Now that we have an element of (4), we can easily turn it into an element of

$$\prod_{x:A} \prod_{C:\prod_{y:A} \mathrm{id}_u(A,x,y)\to\mathcal{U}} C(x,\mathrm{refl}_x) \to \prod_{y:A} \prod_{p:\mathrm{id}_u(A,x,y)} C(y,p),$$

by considering the element as a function taking several arguments and exchanging the order of these arguments. The computation rule is straightforward to show.

Now that we know that id_b and id_u satisfy the exactly the same inference rules, we can conclude that if we prove some statement for id_b, the same statement will also be provable for id_u in place of id_b, and vice versa. Once we have introduced the univalence axiom, we can show that these two type families are actually themselves equal. So there is no need to distinguish them, and in fact we will omit this distinction from now on and simply write $x =_A y$ for either $\mathrm{id}_b(A,x,y)$ or $\mathrm{id}_u(A,x,y)$ and speak of "the" identity type. Since the type A is clear from the context, we will usually just write $x = y$. Our earlier symmetry result can then be written as $(x = y) \to (y = x)$.

Interpretation in terms of paths and homotopies

Finally, we can now explain the "homotopy" in homotopy type theory. While everything else that we have done so far also applies to many other flavours of type theory, this particular behaviour of identity types that we find here is specific to HoTT. In contrast to more conventional type theories, in which the types are interpreted as sets, types in HoTT behave like *spaces* in the sense of homotopy types or, equivalently, as ∞-groupoids. (These two points of view are equivalent by the homotopy hypothesis.)

So we think of any type A as a topological space whose points are the elements $x, y : A$. While this is nothing else than in intuition if one regards HoTT as a foundation of mathematics, there is also a sense in which it can turned into a precise statement inside conventional foundations by finding a *model* of HoTT in conventional foundations. Doing this is important for knowing that if conventional foundations are consistent, then so is HoTT. This is an important result: no contradiction has been found in all of mathematics as based on conventional foundations in close to 100 years. Since HoTT can be modelled inside conventional foundations, this makes it at least as unlikely that HoTT is inconsistent as it is unlikely that conventional foundations are inconsistent.

In any case, in the interpretation that types are spaces and elements of types are points, an element of an identity type $p : x = y$ corresponds to a *path* moving from the point x to the point y—in particular, the letter p actually stands for "path". The identity type $x = y$ does then itself need to be a space, namely the space of all paths from x to y! For paths $p, q : x = y$, the corresponding identity type between p and q then represents the space of all *paths between paths*; commonly, a path between paths is known as a *homotopy* of paths.