Introduction to Type Theories — Anja Petković Komel

Lecture 3 - *June 25, 2025*

The main topic of this lecture is to present the identity type.

In Egbert Rijke's textbook [2], the identity type is presented as one-sided. Here, the identity type will be presented as two-sided.

1 Another Definition of the Induction Principle for Natural Numbers

Last lecture, the inductive principle for natural numbers was defined via Pi types. A definition which does not rely on Pi types is nicer from a proof-theoretic perspective because it allows for a type theory *with* natural numbers, but *without* functions. See the alternative definition below.

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n)\ \text{type} \qquad \Gamma \vdash p_0 : P(0) \qquad \Gamma, n : \mathbb{N}, p : P(n) \vdash p_S(n, p) : P(succ(n)) \qquad \Gamma \vdash k : \mathbb{N}}{\Gamma \vdash \text{ind}_{\mathbb{N}}(P, p_0, p_S, k) : P(k)}$$

2 Two-Side Identity Type

We now turn our attention to the main subject of this lecture: the identity type, also called the propositional equality type. This type is very important because we would like to state that objects are equal, and prove these objects are equal. Thus, we must have propositional equality types (via Curry-Howard). Although we already have an equality judgment, $\Gamma \vdash a \doteq b : A$, this is not a *type*, and having a type to represent equality is a lot more powerful: we'll be able to do far more with it inside the type theory. (Intuitively, the identity type will be able to prove all kinds of non-trivial equalities, while judgmental equality will only be able to talk about simple syntactic rewrites.)

The rules are as follows.

- Formation rule: Given two elements with the same type ($a : A$ and $b : A$), $a =_A b$ is a type. Elements of this type ($p : a =_A b$) describe "the ways that a, b can be equal" (*i.e.* proofs or witnesses). It's an empty type if a and b are different, and nonempty if they're equal.

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : A}{\Gamma \vdash a =_A b \text{ type}}$$

  A concrete example is,

$$\frac{\vdash 0 : \mathbb{N} \qquad \vdash \text{add}_{\mathbb{N}}(0,0) : \mathbb{N}}{\vdash 0 =_{\mathbb{N}} \text{add}_{\mathbb{N}}(0,0) \text{ type}}$$

- Introduction rule: given a single element $a : A$, this gives us a proof of reflexivity, that a equals itself.

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \text{refl}_a : a =_A a}$$

  A concrete example is,

$$\frac{\vdash 0 : \mathbb{N}}{\vdash \text{refl}_0 : 0 =_A 0}$$

  This is slightly less restrictive than it seems at first, since we can always substitute a judgmentally equal thing for one of the a's (using the element conversion rule), deriving the rule

$$\frac{\Gamma \vdash a \doteq b : A}{\Gamma \vdash \text{refl}_a : a =_A b.}$$

- Elimination rule (J-rule): This rule is the induction principle, and defines how we can form functions out of the identity type. In order to form these functions, we only need a term (proof) for *refl*, because that is the only constructor of the identity type. This analogous to the induction principle on $\mathbb{N}$, where we need to handle the two constructors, 0 and successor.

$$\frac{\Gamma, x : A, y : A, p : x =_A y \vdash P(x,y,p) \text{ type}}{\Gamma \vdash J : \left(\Pi_{(x:A)} P(x,x,\text{refl}_x)\right) \to \Pi_{a:A}\Pi_{b:A}\Pi_{q:a=_A b} P(a,b,q)}$$

  Like other induction principles, this rule is usually hidden in proof assistants which let you do pattern matching and implement the rule under the hood.

  A concrete example:

$$\frac{x : \mathbb{N}, y : \mathbb{N}, p : x =_{\mathbb{N}} y \vdash x =_{\mathbb{N}} \text{add}_{\mathbb{N}}(0,y) \text{ type}}{\vdash J : \left(\Pi_{(x:\mathbb{N})} x =_{\mathbb{N}} \text{add}_{\mathbb{N}}(0,x)\right) \to \Pi_{(x:\mathbb{N})}\Pi_{(y:\mathbb{N})}\Pi_{(q:x=_{\mathbb{N}}y)} x =_{\mathbb{N}} \text{add}_{\mathbb{N}}(0,y)}$$

- Computation rule: The J-rule lets you write a function out of an identity type by only considering refl_x. This rule says that when you give this function refl_a, you get what you expect.

$$\frac{\Gamma, x : A, y : A, p : x =_A y \vdash P(x, y, p) \text{ type} \qquad \Gamma \vdash d : \Pi_{x:A} P(x, x, \text{refl}_x) \qquad \Gamma \vdash a : A}{\Gamma \vdash J(d, a, a, \text{refl}_a) \doteq d(a) : P(a, a, \text{refl}_a)}$$

  The d mentioned in this rule is actually referring to "diagonal".

  Diagonal A definition of the *diagonal* of a type is given at page 150 of [2]: a map $\delta_A : A \to A \times A$ given by $\lambda x.(x, x)$, creating a dependent pair. Such definition sort of resonates with what we did in the computation rule (**but is not want we are doing exactly here**).

  Some general ideas behind the computation rule is that, if we apply the elimination rule in a 'base case' under the inductive principle, then we recover the given data, as hinted by [1].

  As a reminder, what *computation rules* are essentially doing is revealing "what happens when applying a function constructed via the **elimination rule** to an element **constructed via the introduction rule**." [1]

  Example To give a concrete, though not very motivating, example, let's define $d :\equiv \lambda x.\text{refl}_x : \Pi_{(x:\mathbb{N})} x =_{\mathbb{N}} \text{add}(x, 0)$. (Note that this is well-typed by the element conversion rule, since the judgmental equality $x : \mathbb{N} \vdash x \doteq \text{add}(x, 0) : \mathbb{N}$ holds.) Now, as in the J example above, we get $J(d) : \Pi_{x:\mathbb{N}} \Pi_{y:\mathbb{N}} \Pi_{p:x=_{\mathbb{N}} y}\ x =_{\mathbb{N}} \text{add}_{\mathbb{N}}(y, 0)$. If we then apply it to n, n, and refl_n for some natural number $\vdash n : \mathbb{N}$, we have the reduction

$$\vdash J(d, n, n, \text{refl}_n) \doteq d(n) \doteq \text{refl}_n : n =_{\mathbb{N}} \text{add}_{\mathbb{N}}(n, 0).$$

- Congruence rule:

$$\frac{\Gamma \vdash a \doteq a' : A \qquad \Gamma \vdash b \doteq b' : A}{\Gamma \vdash (a =_A b) \doteq (a' =_A b') \text{ type}}$$

  For a concrete example that uses the congruence rule, we would like to prove the following:

$$\frac{\Gamma \vdash a \doteq a' : A \qquad \Gamma \vdash p : a =_A b}{\Gamma \vdash p : a' =_A b}$$

  This shows that if we have a proof $(p : a =_A b)$, and an a' that is judgmentally equal to a, the same proof, p, works for a' and b. Below is the derivation:

$$\dfrac{\dfrac{\Gamma \vdash a \doteq a' : A \qquad \dfrac{\Gamma \vdash b : A}{\Gamma \vdash b \doteq b : A}\ \text{REFL OF JUDGMENT EQ}}{\Gamma \vdash (a =_A b) \doteq (a' =_A b)\ \text{type}}\ \text{CONG} \qquad \Gamma \vdash p : a =_A b}{\Gamma \vdash p : a' =_A b}\ \text{ELEM CONV}$$

Now let's turn to doing some proofs with equality.

Equality is *symmetric*: we'll define a function

$$\text{symm}_A : \Pi_{x:A}\Pi_{y:A}(x =_A y) \to (y =_A x).$$

To do this using J directly, we'll first need a type family:

$$\vdash P(x, y, p) \coloneqq (y =_A x)\ \text{type}$$

Now we can use J to define

$$\text{symm}_A \coloneqq J_P(\lambda x.\text{refl}_x).$$

This is rather cumbersome, so we'll use a pattern matching notation in the future:

$$\text{symm}_A(x, x, \text{refl}_x) \coloneqq \text{refl}_x.$$

Defining $f(x, x, \text{refl}_x)$ will be shorthand for defining f to use the J rule.

Equality is *transitive*:

$$\text{concat}_A : \Pi_{x,y,z:A}(x =_A y) \to (y =_A z) \to (x =_A z)$$
$$\text{concat}_A(x, x, z, \text{refl}_x, p) \coloneqq p$$

We could think of this as "concatenating" a proof connecting x to y and a proof connecting y to z.

Functions are *congruences* for equality:

$$\text{ap}_{A,B} : \Pi_{f:A\to B}\Pi_{x,y:A}(x =_A y) \to (f(x) =_B f(y))$$
$$\text{ap}_{A,B}(f, x, x, \text{refl}_x) \coloneqq \text{refl}_{f(x)}$$

The name comes from the *a*ction on *p*aths of the function f: it takes a path (proof of equality) between x and y, to a path (proof of equality) between $f(x)$ and $f(y)$.

Transport Finally, if $x = y$, then you can turn elements of $B(x)$ into elements of $B(y)$:

$$\mathrm{tr}_{A,B} : \Pi_{x,y:A}(x =_A y) \to B(x) \to B(y)$$
$$\mathrm{tr}_{A,B}(x, x, \mathrm{refl}_x, b) \coloneqq b$$

This is called *tr*ansport, since we move the element b from type $B(x)$ to $B(y)$.

References

[1] Cecilia Flori and Tobias Fritz. *Homotopy Type Theory: Univalent Foundations of Mathematics*. University of Waterloo Graduate Course Notes, Chapter 6. Spring 2014, Perimeter Institute for Theoretical Physics. 2014. URL: `http://tobiasfritz.science/2014/topic6.pdf`.

[2] Egbert Rijke. *Introduction to Homotopy Type Theory*. 2022. arXiv: `2212.11082 [math.LO]`. URL: `https://arxiv.org/abs/2212.11082`.