Exercises – Introduction to Type Theory

Exercise 4 – Dependent Types

1. The classical statement of Fermat's last theorem is

   $$\textit{There are no three positive integers } a, b, c \textit{ such that } a^n + b^n = c^n \textit{ when } n \geq 3.$$

   Write down a type which encodes the statement of Fermat's last theorem.

2. Assume that A and B are types and $R : A \to B \to \mathsf{Type}$ is a relation. Find a term t which admits the following type:

   $$\big(\Pi\,(x : A)\,.\,\Sigma\,(y : B)\,.\,R\ x\ y\big) \to \big(\Sigma\,(f : A \to B)\,.\,\Pi\,(x : A)\,.\,R\ x\ (f\ x)\big)$$

   To draw a parallel with classical mathematics, replace Π with $\forall$ and Σ with $\exists$. Do you recognise the statement that you just proved?

Solution

1. We are going to assume that we have access to the following two terms:

   - $\mathsf{plus} : \mathbb{N} \to \mathbb{N}$ which computes the addition of two natural numbers,
   - $\mathsf{pow} : \mathbb{N} \to \mathbb{N}$ such that $\mathsf{pow}\ a\ b$ computes a^b.

   An explicit definition of plus will be given in exercise 5. Using these two functions, we can encode the statement of Fermat's last theorem with the following type:

   $$\Pi\,(a\ b\ c\ n : \mathbb{N})\,.\,\Big((a+1)^{(n+3)} + (b+1)^{(n+3)} =_{\mathbb{N}} (c+1)^{(n+3)}\Big) \to \bot$$

   Note that we used a few notational shortcuts for conciseness. A purely formal term would use plus and pow instead of the usual mathematical notation, and four separate quantifiers:

   $$\Pi\,(a : \mathbb{N})\,.\,\Pi\,(b : \mathbb{N})\,.\,\Pi\,(c : \mathbb{N})\,.\,\Pi\,(n : \mathbb{N})\,.$$
   $$(\mathsf{plus}\ (\mathsf{pow}\ (\mathsf{plus}\ a\ 1)\ (\mathsf{plus}\ n\ 3))\ (\mathsf{pow}\ (\mathsf{plus}\ b\ 1)\ (\mathsf{plus}\ n\ 3)) =_{\mathbb{N}} \mathsf{pow}\ (\mathsf{plus}\ c\ 1)\ (\mathsf{plus}\ n\ 3)) \to \bot$$

2. The following term works: $\lambda\,f\,.\,\langle\,\lambda\,(x : A)\,.\,\pi_1\ (f\ x)\,,\ \lambda\,(x : A)\,.\,\pi_2\ (f\ x)\,\rangle$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f : [...], x : A \vdash f : \Pi\,x\,.\,\Sigma\,y\,.\,R\ x\ y} \quad \dfrac{}{f : [...], x : A \vdash x : A}}{f : [...], x : A \vdash f\ x : \Sigma\,(y : B)\,.\,R\ x\ y}}{f : [...], x : A \vdash \pi_1\ (f\ x) : B}}{f : \Pi\,x\,.\,\Sigma\,y\,.\,R\ x\ y \vdash \lambda\,x\,.\,\pi_1\ (f\ x) : A \to B} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{f : [...], x : A \vdash f : \Pi\,x\,.\,\Sigma\,y\,.\,R\ x\ y} \quad \dfrac{}{f : [...], x : A \vdash x : A}}{f : [...], x : A \vdash f\ x : \Sigma\,(y : B)\,.\,R\ x\ y}}{f : [...], x : A \vdash \pi_2\ (f\ x) : R\ x\ (\pi_1\ (f\ x))}}{f : \Pi\,x\,.\,\Sigma\,y\,.\,R\ x\ y \vdash \lambda\,x\,.\,\pi_2\ (f\ x) : \Pi\,x\,.\,R\ x\ (\pi_1\ (f\ x))}}{f : \Pi\,x\,.\,\Sigma\,y\,.\,R\ x\ y \vdash \langle\,\lambda\,x\,.\,\pi_1\ (f\ x)\,,\ \lambda\,x\,.\,\pi_2\ (f\ x)\,\rangle : \Sigma\,f\,.\,\Pi\,x\,.\,R\ x\ (f\ x)}$$
$$\vdash \lambda\,f\,.\,\langle\,\lambda\,x\,.\,\pi_1\ (f\ x)\,,\ \lambda\,x\,.\,\pi_2\ (f\ x)\,\rangle : (\Pi\,x\,.\,\Sigma\,y\,.\,R\ x\ y) \to (\Sigma\,f\,.\,\Pi\,x\,.\,R\ x\ (f\ x))$$

Note that in the above tree, we crucially rely on the *conversion rule*, which lets us unfold definitional equations inside of types.

If we replace Π with $\forall$ and Σ with $\exists$, the statement that we just proved is exactly the axiom of choice. Somehow, it seems that the axiom of choice is *provable* in dependent type theory, even though dependent type theory is constructive (unless we explicitly assume non-constructive principles)! This stems from the fact that Σ is not exactly the same thing as an existential quantifier: when we define a term of type $\Sigma\,(x : A)\,.\,P\ x$, the term *remembers* the specific element of A which witnesses the property P, and we can extract it using the first projection.

Exercise 5 – Natural numbers

1. Use the recursor of $\mathbb{N}$ to define addition as a function $\mathsf{plus} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$.
2. Use the recursor of $\mathbb{N}$ to define a binary relation $\mathsf{gt} : \mathbb{N} \to \mathbb{N} \to \mathsf{Type}$ which encodes the (strict) order relation on $\mathbb{N}$. Can you think of another definition?
3. Find a proof term for the statement $\Pi\,(n : \mathbb{N})\,.\,\mathsf{gt}\ (\mathsf{plus}\ 1\ n)\ n$.

Solution

1. The addition is defined by the following recursive equations:

$$\begin{aligned}&\mathsf{plus} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})\\ &\mathsf{plus}\ n\ 0 \qquad\quad \equiv n\\ &\mathsf{plus}\ n\ (\mathsf{suc}\ m) \equiv \mathsf{suc}\ (\mathsf{plus}\ n\ m)\end{aligned}$$

   We note that the partially applied term $(\mathsf{plus}\ n)$ is a function of type $\mathbb{N} \to \mathbb{N}$ defined by a simple recursion. We can translate this definition to one which uses the eliminator $\mathbb{N}{-}\mathsf{ind}$ as follows:

   - the dependent codomain is $\lambda\, m\,.\,\mathbb{N}$
   - the image of $m = 0$ is n
   - the function which goes from the image of m to the image of $m + 1$ is $\lambda\, m\,.\,\lambda\, x\,.\,\mathsf{suc}\ x$

   Thus the term is $\mathsf{plus} \equiv \lambda\,(n : \mathbb{N})\,.\,\lambda\,(m : \mathbb{N})\,.\,\mathbb{N}{-}\mathsf{ind}(\lambda\, m\,.\,\mathbb{N},\ \ n,\ \ \lambda\, m\,.\,\lambda\, x\,.\,\mathsf{suc}\ x,\ \ m)$.

2. The order relation is defined by the following recursive equations:

$$\begin{aligned}&\mathsf{gt} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})\\ &\mathsf{gt}\ 0\ m \qquad\quad\ \equiv \bot\\ &\mathsf{gt}\ (\mathsf{suc}\ n)\ 0 \qquad \equiv \top\\ &\mathsf{gt}\ (\mathsf{suc}\ n)\ (\mathsf{suc}\ m) \equiv \mathsf{gt}\ n\ m\end{aligned}$$

   Rephrasing this definition in terms of the eliminator $\mathbb{N}{-}\mathsf{ind}$ is a bit more difficult than plus, because it involves a case analysis on both n and m, while the eliminator $\mathbb{N}{-}\mathsf{ind}$ lets us do induction on a single variable of type $\mathbb{N}$.

   The trick is to define the *partially applied function* gt by using induction on the first variable. We can define $\mathsf{gt}\ 0$ as the constant function $\lambda\, m\,.\,\bot$, and assuming that $\mathsf{gt}\ n$ is defined as an element of $\mathbb{N} \to \mathbb{N}$, we can define $\mathsf{gt}\ (\mathsf{suc}\ n)$ by using another induction on the second argument. Formally, this becomes

$$\mathsf{gt} \equiv \lambda\,(n : \mathbb{N})\,.\,\mathbb{N}{-}\mathsf{ind}\big(\lambda\, n\,.\,\mathbb{N} \to \mathbb{N}, \lambda\, m\,.\,\bot, \lambda\, n\,.\,\lambda\, f\,.\,(\lambda\, m\,.\,\mathbb{N}{-}\mathsf{ind}(\lambda\, m\,.\,\mathbb{N}, \top, \lambda\, m\,.\,\lambda\, x\,.\,f\ m)), n\big)$$

   As you can see, we can systematically translate definitions by recursive equations into definitions based on $\mathbb{N}{-}\mathsf{ind}$, but the result quickly becomes unreadable. In practice, we always use definitions by recursive equations, and we leave the translation to a proof assistant.

3. The statement that we want to prove is saying that for all $n : \mathbb{N}$, we have $1 + n > n$. We will prove this by induction (which really amounts to using the $\mathbb{N}-\mathsf{ind}$ principle).

   To start our induction, we need to prove the base case, *i.e.*, we need to supply an element of the type $\mathsf{gt}\ (\mathsf{plus}\ 1\ 0)\ 0$. We can simplify this type by unfolding the equations that define plus:

$$\begin{aligned} \mathsf{plus}\ 1\ 0 &\equiv \mathsf{plus}\ (\mathsf{suc}\ 0)\ 0 \\ &\equiv \mathsf{suc}\ (\mathsf{plus}\ 0\ 0) \\ &\equiv \mathsf{suc}\ 0 \end{aligned}$$

   Therefore, the base case is equivalently asking us to give a term of type $\mathsf{gt}\ (\mathsf{suc}\ 0)\ 0$. Using the defining equations of gt, we can simplify this type even further:

$$\mathsf{gt}\ (\mathsf{suc}\ 0)\ 0 \equiv \top$$

   Thus the base case simply amounts to finding a term of type $\top$, for which we can use $\star$.

   Now for the induction step. We assume that we have a term of type $h : \mathsf{gt}\ (\mathsf{plus}\ 1\ n)\ n$, and we need to construct a term of type $\mathsf{gt}\ (\mathsf{plus}\ 1\ (\mathsf{suc}\ n))\ (\mathsf{suc}\ n)$. We first simplify the subterm $\mathsf{plus}\ 1\ (\mathsf{suc}\ n)$ to obtain $\mathsf{suc}\ (\mathsf{plus}\ 1\ n)$. Next, we can simplify $\mathsf{gt}\ (\mathsf{suc}\ (\mathsf{plus}\ 1\ n))\ (\mathsf{suc}\ n)$ to obtain $\mathsf{gt}\ (\mathsf{plus}\ 1\ n)\ n$, which is exactly the type of h, so we can use the induction hypothesis as is.

   Putting everything together, we obtain the following proof:

$$\begin{aligned} &f : \Pi\,(n : \mathbb{N})\,.\,\mathsf{gt}\ (\mathsf{plus}\ 1\ n)\ n \\ &f\ 0 \equiv \star \\ &f\ (\mathsf{suc}\ n) \equiv f\ n \end{aligned}$$

Exercise 6 – Martin-Löf's equality type

Assume that A is a type, and that x, y, z are three terms of type A.

1. Find a proof term for the statement $(x = y) \to (y = x)$.
2. Find a proof term for the statement $(x = y) \to (y = z) \to (x = z)$.
3. ($\star$) Find a proof term for $\Pi\,(n : \mathbb{N})\,.\,\Pi\,(m : \mathbb{N})\,.\,\mathsf{plus}\ n\ m = \mathsf{plus}\ m\ n$.
   Hint: you might want to define an auxiliary lemma.

Solution

1. The elimination rule for equality (also known as *path induction*) is telling us that if we have a type $B(x, e)$ which depends on a variable $x : A$ and a variable $e : t =_A x$, then proving $B(x, e)$ for any particular instance of x and e can be reduced to proving $B(t, \mathsf{refl})$.

   Now, assume that we have a term $e : x =_A y$. Our goal is to construct a term of type $B(y, e) \equiv (y =_A x)$. We can use path induction to reduce the goal to $B(x, \mathsf{refl}) \equiv (x =_A x)$, which is provable by refl.

   This reasoning can be formalised as the term $\lambda\,(e : x =_A y)\,.\,\mathsf{J}\big(A, x, (\lambda\,y\,.\,\lambda\,e\,.\,y =_A x), \mathsf{refl}, y, e\big)$.

2. Assume $e : x =_A y$ and $e' : y =_A z$. Our goal is to construct a term of type $B(z, e') \equiv x =_A z$. We use path induction to reduce this goal to $B(y, \mathsf{refl}) \equiv (x =_A y)$, which is provable by e.

   This corresponds to the term $\lambda\,(e : x =_A y)\,.\,\lambda\,(e' : y =_A z)\,.\,\mathsf{J}\big(A, y, (\lambda\,z\,.\,\lambda\,e'\,.\,x =_A z), e, z, e'\big)$.

3. We start by constructing a term $\mathsf{plus0}$ of type $\Pi\,(n : \mathbb{N})\,.\,\mathsf{plus}\ 0\ n =_{\mathbb{N}} n$, by induction:

   – The base case is $(\mathsf{plus}\ 0\ 0) =_{\mathbb{N}} 0$, which reduces to $0 =_{\mathbb{N}} 0$. This can be proved by refl.

   – For the induction step, assume that we have a term $e : \mathsf{plus}\ 0\ n =_{\mathbb{N}} n$. We need to prove $\mathsf{plus}\ 0\ (\mathsf{suc}\ n) =_{\mathbb{N}} \mathsf{suc}\ n$, which reduces to $\mathsf{suc}\ (\mathsf{plus}\ 0\ n) =_{\mathbb{N}} \mathsf{suc}\ n$. We can do this using $\mathsf{ap}\ \mathsf{suc}\ e$.

   Next, we construct a term $\mathsf{plusSuc}$ of type $\Pi\,(n : \mathbb{N})\,.\,\Pi\,(m : \mathbb{N})\,.\,\mathsf{plus}\ (\mathsf{suc}\ n)\ m =_{\mathbb{N}} \mathsf{suc}\,(\mathsf{plus}\ n\ m)$, by induction on m:

   – The base case is $(\mathsf{plus}\ (\mathsf{suc}\ n)\ 0) =_{\mathbb{N}} \mathsf{suc}\ n$, which reduces to $\mathsf{suc}\ n =_{\mathbb{N}} \mathsf{suc}\ n$. This can be proved by refl.

   – For the induction step, assume that we have a term $e : \mathsf{plus}\ (\mathsf{suc}\ n)\ m =_{\mathbb{N}} \mathsf{suc}\,(\mathsf{plus}\ n\ m)$. We need to construct a term of type $\mathsf{plus}\ (\mathsf{suc}\ n)\ (\mathsf{suc}\ m) =_{\mathbb{N}} \mathsf{suc}\,(\mathsf{plus}\ n\ (\mathsf{suc}\ m))$, which reduces to $\mathsf{suc}\ (\mathsf{plus}\ (\mathsf{suc}\ n)\ m) =_{\mathbb{N}} \mathsf{suc}\ (\mathsf{suc}\,(\mathsf{plus}\ n\ m))$. We can do this using $\mathsf{ap}\ \mathsf{suc}\ e$.

   Now that we have our two lemmas, we prove that $\mathsf{plus}\ n\ m = \mathsf{plus}\ m\ n$ by induction on n. Simplifying the equations reveals that the base case is exactly $\mathsf{plus0}$, and the induction step is exactly $\mathsf{plusSuc}$.