

# Exercises from the *HoTT Book*

John Dougherty

June 30, 2014

## Introduction

The following are solutions to (eventually all of) the exercises from *Homotopy Type Theory: Univalent Foundations of Mathematics*. The Coq code given alongside the by-hand solutions requires the HoTT version of Coq, available [at the HoTT github repository](#). It will be assumed throughout that it has been imported by

`Require Import HoTT.`

The [introduction to Coq from the HoTT repo](#) is assumed. Each exercise has its own `Section` in the Coq file, so `Context` declarations don't extend beyond the exercise—and sometimes they're even more restricted than that.

## 1 Type Theory

**Exercise 1.1 (p. 56)** Given functions  $f : A \rightarrow B$  and  $g : B \rightarrow C$ , define their **composite**  $g \circ f : A \rightarrow C$ . Show that we have  $h \circ (g \circ f) \equiv (h \circ g) \circ f$ .

**Solution** Define  $g \circ f := \lambda(x : A). g(f(x))$ . Then if  $h : C \rightarrow D$ , we have

$$h \circ (g \circ f) \equiv \lambda(x : A). h((g \circ f)x) \equiv \lambda(x : A). h((\lambda(y : A). g(fy))x) \equiv \lambda(x : A). h(g(fx))$$

and

$$(h \circ g) \circ f \equiv \lambda(x : A). (h \circ g)(fx) \equiv \lambda(x : A). (\lambda(y : A). h(gy))(fx) \equiv \lambda(x : A). h(g(fx))$$

So  $h \circ (g \circ f) \equiv (h \circ g) \circ f$ . In Coq, we have

**Definition** `compose`  $\{A B C : \text{Type}\} (g : B \rightarrow C) (f : A \rightarrow B) := \text{fun } x \Rightarrow g (f x)$ .

**Theorem** `compose_assoc`  $\forall (A B C D : \text{Type}) (f : A \rightarrow B) (g : B \rightarrow C) (h : C \rightarrow D)$ ,  
`compose h (compose g f) = compose (compose h g) f`.

**Exercise 1.2 (p. 56)** Derive the recursion principle for products  $\text{rec}_{A \times B}$  using only the projections, and verify that the definitional equalities are valid. Do the same for  $\Sigma$ -types.

**Section** `Exercise2a`.

**Context**  $\{A B : \text{Type}\}$ .

**Solution** The recursion principle states that we can define a function  $f : A \times B \rightarrow C$  by giving its value on pairs. Suppose that we have projection functions  $\text{pr}_1 : A \times B \rightarrow A$  and  $\text{pr}_2 : A \times B \rightarrow B$ . Then we can define a function of type

$$\text{rec}_{A \times B} : \prod_{C : \mathcal{U}} (A \rightarrow B \rightarrow C) \rightarrow A \times B \rightarrow C$$

in terms of these projections as follows

$$\text{rec}'_{A \times B}(C, g, p) := g(\text{pr}_1 p)(\text{pr}_2 p)$$

or, in Coq,

**Definition** `recprod` ( $C : \text{Type}$ ) ( $g : A \rightarrow B \rightarrow C$ ) ( $p : A \times B$ ) :=  $g (\text{fst } p) (\text{snd } p)$ .

We must then show that

$$\text{rec}'_{A \times B}(C, g, (a, b)) \equiv g(\text{pr}_1(a, b))(\text{pr}_2(a, b)) \equiv g(a)(b)$$

which in Coq is also trivial:

**Goal**  $\forall C g a b, \text{recprod } C g (a, b) = g a b$ .

Now for the  $\Sigma$ -types. Here we have a projection

$$\text{pr}_1 : \left( \sum_{x:A} B(x) \right) \rightarrow A$$

and another

$$\text{pr}_2 : \prod_{p: \sum_{(x:A)} B(x)} B(\text{pr}_1(p))$$

Define a function of type

$$\text{rec}_{\sum_{(x:A)} B(x)} : \prod_{C:\mathcal{U}} \left( \prod_{(x:A)} B(x) \rightarrow C \right) \rightarrow \left( \sum_{(x:A)} B(x) \right) \rightarrow C$$

by

$$\text{rec}_{\sum_{(x:A)} B(x)}(C, g, p) := g(\text{pr}_1 p)(\text{pr}_2 p)$$

**Definition** `recsm` ( $C : \text{Type}$ ) ( $g : \forall (x : A), B x \rightarrow C$ ) ( $p : \exists (x : A), B x$ ) :=  $g (\text{projT1 } p) (\text{projT2 } p)$ .

We then verify that

$$\text{rec}_{\sum_{(x:A)} B(x)}(C, g, (a, b)) \equiv g(\text{pr}_1(a, b))(\text{pr}_2(a, b)) \equiv g(a)(b)$$

which is again trivial in Coq:

**Goal**  $\forall C g a b, \text{recsm } C g (a; b) = g a b$ .

**Exercise 1.3 (p. 56)** Derive the induction principle for products  $\text{ind}_{A \times B}$  using only the projections and the propositional uniqueness principle uppt. Verify that the definitional equalities are valid. Generalize uppt to  $\Sigma$ -types, and do the same for  $\Sigma$ -types.

**Solution** The induction principle has type

$$\text{ind}_{A \times B} : \prod_{C:A \times B \rightarrow \mathcal{U}} \left( \prod_{(x:A)} \prod_{(y:B)} C((x, y)) \right) \rightarrow \prod_{z:A \times B} C(z)$$

For a first pass, we can define

$$\text{ind}_{A \times B}(C, g, z) := g(\text{pr}_1 z)(\text{pr}_2 z)$$

However, we have  $g(\text{pr}_1 x)(\text{pr}_1 x) : C((\text{pr}_1 x, \text{pr}_2 x))$ , so the type of this  $\text{ind}_{A \times B}$  is

$$\text{ind}_{A \times B} : \prod_{C : A \times B \rightarrow \mathcal{U}} \left( \prod_{(x:A)} \prod_{(y:B)} C((x, y)) \right) \rightarrow \prod_{z : A \times B} C((\text{pr}_1 z, \text{pr}_2 z))$$

To define  $\text{ind}_{A \times B}$  with the correct type, we need the transport operation from the next chapter. The uniqueness principle for  $A \times B$  is

$$\text{uppt} : \prod_{x : A \times B} ((\text{pr}_1 x, \text{pr}_2 x) =_{A \times B} x)$$

By the transport principle, there is a function

$$(\text{uppt } x)_* : C((\text{pr}_1 x, \text{pr}_2 x)) \rightarrow C(x)$$

so

$$\text{ind}_{A \times B}(C, g, z) \equiv (\text{uppt } z)_*(g(\text{pr}_1 z)(\text{pr}_2 z))$$

has the right type. In Coq we first define `uppt`, then use it with `transport` to give our  $\text{ind}_{A \times B}$ .

**Definition** `uppt`  $(x : A \times B) : (\text{fst } x, \text{snd } x) = x$ .

**Definition** `indprd`  $(C : A \times B \rightarrow \text{Type}) (g : \forall (x:A) (y:B), C(x, y)) (z : A \times B) :=$   
 $(\text{uppt } z) \# (g(\text{fst } z)(\text{snd } z))$ .

We now have to show that

$$\text{ind}_{A \times B}(C, g, (a, b)) \equiv g(a)(b)$$

Unfolding the left gives

$$\begin{aligned} \text{ind}_{A \times B}(C, g, (a, b)) &\equiv (\text{uppt } (a, b))_*(g(\text{pr}_1(a, b))(\text{pr}_2(a, b))) \\ &\equiv \text{ind}_{=_{A \times B}}(D, d, (a, b), (a, b), \text{uppt}((a, b)))(g(a)(b)) \\ &\equiv \text{ind}_{=_{A \times B}}(D, d, (a, b), (a, b), \text{refl}_{(a, b)})(g(a)(b)) \\ &\equiv \text{ind}_{=_{A \times B}}(D, d, (a, b), (a, b), \text{refl}_{(a, b)})(g(a)(b)) \\ &\equiv \text{id}_{C((a, b))}(g(a)(b)) \\ &\equiv g(a)(b) \end{aligned}$$

which was to be proved. In Coq, it's as trivial as always:

**Goal**  $\forall C g a b, \text{indprd } C g (a, b) = g a b$ .

For  $\Sigma$ -types, we define

$$\text{ind}_{\Sigma_{(x:A)} B(x)} : \prod_{C : (\Sigma_{(x:A)} B(x)) \rightarrow \mathcal{U}} \left( \prod_{(a:A)} \prod_{(b:B(a))} C((a, b)) \right) \rightarrow \prod_{p : \Sigma_{(x:A)} B(x)} C(p)$$

at first pass by

$$\text{ind}_{\Sigma_{(x:A)} B(x)}(C, g, p) \equiv g(\text{pr}_1 p)(\text{pr}_2 p)$$

We encounter a similar problem as before. We need a uniqueness principle for  $\Sigma$ -types, which would be a function

$$\text{upst} : \prod_{p : \Sigma_{(x:A)} B(x)} ((\text{pr}_1 p, \text{pr}_2 p) =_{\Sigma_{(x:A)} B(x)} p)$$

As for product types, we can define

$$\text{upst}((a, b)) \equiv \text{refl}_{(a, b)}$$

which is well-typed, since  $\text{pr}_1(a, b) \equiv a$  and  $\text{pr}_2(a, b) \equiv b$ . Thus, we can write

$$\text{ind}_{\Sigma_{(x:A)} B(x)}(C, g, p) \equiv (\text{upst } p)_*(g(\text{pr}_1 p)(\text{pr}_2 p)).$$

and in Coq,

**Definition**  $\text{upst}(p : \{x:A \& B\ x\}) : (\text{projT1 } p; \text{projT2 } p) = p.$

**Definition**  $\text{indsm}(C : \{x:A \& B\ x\} \rightarrow \text{Type}) (g : \forall (a:A) (b:B\ a), C\ (a; b)) (p : \{x:A \& B\ x\}) :=$   
 $(\text{upst } p) \# (g (\text{projT1 } p) (\text{projT2 } p)).$

Now we must verify that

$$\text{ind}_{\Sigma_{(x:A)} B(x)}(C, g, (a, b)) \equiv g(a)(b)$$

We have

$$\begin{aligned} \text{ind}_{\Sigma_{(x:A)} B(x)}(C, g, (a, b)) &\equiv (\text{upst } (a, b))_*(g(\text{pr}_1(a, b))(\text{pr}_2(a, b))) \\ &\equiv \text{ind}_{=\Sigma_{(x:A)} B(x)}(D, d, (a, b), (a, b), \text{upst } (a, b))(g(a)(b)) \\ &\equiv \text{ind}_{=\Sigma_{(x:A)} B(x)}(D, d, (a, b), (a, b), \text{refl}_{(a, b)})(g(a)(b)) \\ &\equiv \text{id}_{C((a, b))}(g(a)(b)) \\ &\equiv g(a)(b) \end{aligned}$$

which Coq finds trivial:

**Goal**  $\forall C\ g\ a\ b, \text{indsm } C\ g\ (a; b) = g\ a\ b.$

**Exercise 1.4 (p. 56)** Assuming as given only the *iterator* for natural numbers

$$\text{iter} : \prod_{C:\mathcal{U}} C \rightarrow (C \rightarrow C) \rightarrow \mathbb{N} \rightarrow C$$

with the defining equations

$$\begin{aligned} \text{iter}(C, c_0, c_s, 0) &\equiv c_0, \\ \text{iter}(C, c_0, c_s, \text{succ}(n)) &\equiv c_s(\text{iter}(C, c_0, c_s, n)), \end{aligned}$$

derive a function having the type of the recursor  $\text{rec}_{\mathbb{N}}$ . Show that the defining equations of the recursor hold propositionally for this function, using the induction principle for  $\mathbb{N}$ .

**Solution** Fix some  $C : \mathcal{U}$ ,  $c_0 : C$ , and  $c_s : \mathbb{N} \rightarrow C \rightarrow C$ .  $\text{iter}(C)$  allows for the  $n$ -fold application of a single function to a single input from  $C$ , whereas  $\text{rec}_{\mathbb{N}}$  allows each application to depend on  $n$ , as well. Since  $n$  just tracks how many applications we've done, we can construct  $n$  on the fly, iterating over elements of  $\mathbb{N} \times C$ . So we will use the iterator

$$\text{iter}_{\mathbb{N} \times C} : \mathbb{N} \times C \rightarrow (\mathbb{N} \times C \rightarrow \mathbb{N} \times C) \rightarrow \mathbb{N} \rightarrow \mathbb{N} \times C$$

to derive a function

$$\Phi : \prod_{C:\mathcal{U}} C \rightarrow (\mathbb{N} \rightarrow C \rightarrow C) \rightarrow \mathbb{N} \rightarrow C$$

which has the same type as  $\text{rec}_{\mathbb{N}}$ .

The first argument of  $\text{iter}_{\mathbb{N} \times C}$  is the starting point, which we'll make  $(0, c_0)$ . The second input takes an element of  $\mathbb{N} \times C$  as an argument and uses  $c_s$  to construct a new element of  $\mathbb{N} \times C$ . We can use the first and second elements of the pair as arguments for  $c_s$ , and we'll use  $\text{succ}$  to advance the second argument, representing the number of steps taken. This gives the function

$$\lambda x. (\text{succ}(\text{pr}_1 x), c_s(\text{pr}_1 x, \text{pr}_2 x)) : \mathbb{N} \times C \rightarrow \mathbb{N} \times C$$

for the second input to  $\text{iter}_{\mathbb{N} \times C}$ . The third input is just  $n$ , which we can pass through. Plugging these in gives

$$\text{iter}_{\mathbb{N} \times C}((0, c_0), \lambda x. (\text{succ}(\text{pr}_1 x), c_s(\text{pr}_1 x, \text{pr}_2 x)), n) : \mathbb{N} \times C$$

from which we need to extract an element of  $C$ . This is easily done with the projection operator, so we have

$$\Phi_C(c_0, c_s, n) \equiv \text{pr}_2 \left( \text{iter}_{\mathbb{N} \times C}((0, c_0), \lambda x. (\text{succ}(\text{pr}_1 x), c_s(\text{pr}_1 x, \text{pr}_2 x)), n) \right)$$

which has the same type as  $\text{rec}_{\mathbb{N}}$ . In Coq we first define the iterator and then our alternative recursor:

```
Fixpoint iter (C : Type) (c0 : C) (cs : C → C) (n : nat) : C :=
  match n with
  | 0 ⇒ c0
  | S n ⇒ cs (iter C c0 cs n)
  end.

Definition Phi (C : Type) (c0 : C) (cs : nat → C → C) (n : nat) :=
  snd (iter (nat × C)
    (0, c0)
    (fun x ⇒ (S (fst x), cs (fst x) (snd x)))
    n).
```

Now to show that the defining equations hold propositionally for  $\Phi$ . To do this, we must show that

$$\begin{aligned} \Phi(C, c_0, c_s, 0) &=_{\text{C}} c_0 \\ \prod_{n:\mathbb{N}} \left( \Phi(C, c_0, c_s, \text{succ}(n)) &=_{\text{C}} c_s(n, \Phi(C, c_0, c_s, n)) \right) \end{aligned}$$

are inhabited. Since  $C$ ,  $c_0$ , and  $c_s$  are fixed, define for brevity. The first equality is straightforward:

$$\Phi(C, c_0, c_s, 0) \equiv \text{pr}_2 \left( \text{iter}_{\mathbb{N} \times C}((0, c_0), \lambda x. (\text{succ}(\text{pr}_1 x), c_s(\text{pr}_1 x, \text{pr}_2 x)), 0) \right) \equiv \text{pr}_2(0, c_0) \equiv c_0$$

and in Coq,

```
Goal ∀ C c0 cs, Phi C c0 cs 0 = c0.
```

So  $\text{refl}_{c_0} : \Phi(C, c_0, c_s, 0) =_{\text{C}} c_0$ . This establishes the first equality. We prove the second by strengthening the induction hypothesis. Define  $\Phi'$  as the argument of  $\text{pr}_2$  in the above definition; i.e., such that  $\Phi = \text{pr}_2 \Phi'$ .

```
Definition Phi' (C : Type) (c0 : C) (cs : nat → C → C) (n : nat) :=
  iter (nat × C) (0, c0) (fun x ⇒ (S (fst x), cs (fst x) (snd x))) n.
```

We then show that for all  $n : \mathbb{N}$ ,

$$P(n) \equiv (\Phi'(C, c_0, c_s, \text{succ}(n)) =_{\text{C}} (n, c_s(n, \text{pr}_2 \Phi'(C, c_0, c_s, n)))) .$$

For the base case, consider  $\Phi'(C, c_0, c_s, 0)$ ; we have

$$\begin{aligned}\Phi'(C, c_0, c_s, \text{succ}(0)) &\equiv \text{iter}_{\mathbb{N} \times C}((0, c_0), \lambda x. (\text{succ}(\text{pr}_1 x), c_s(\text{pr}_1 x, \text{pr}_2 x)), \text{succ}(0)) \\ &\equiv (\lambda x. (\text{succ}(\text{pr}_1 x), c_s(\text{pr}_1 x, \text{pr}_2 x))) \Phi'(C, c_0, c_s, 0) \\ &\equiv (\text{succ}(\text{pr}_1(0, c_0)), c_s(\text{pr}_1(0, c_0), \text{pr}_2 \Phi'(C, c_0, c_s, 0))) \\ &\equiv (\text{succ}(0), c_s(0, \text{pr}_2 \Phi'(C, c_0, c_s, 0)))\end{aligned}$$

For the induction step, suppose that  $n : \mathbb{N}$  and that  $P(n)$  is inhabited. Then

$$\begin{aligned}\Phi'(C, c_0, c_s, \text{succ}(\text{succ}(n))) &\equiv \text{iter}_{\mathbb{N} \times C}((0, c_0), \lambda x. (\text{succ}(\text{pr}_1 x), c_s(\text{pr}_1 x, \text{pr}_2 x)), \text{succ}(\text{succ}(n))) \\ &\equiv (\lambda x. (\text{succ}(\text{pr}_1 x), c_s(\text{pr}_1 x, \text{pr}_2 x))) \Phi'(C, c_0, c_s, \text{succ}(n)) \\ &\equiv (\text{succ}(\text{pr}_1 \Phi'(C, c_0, c_s, \text{succ}(n))), \\ &\quad c_s(\text{pr}_1 \Phi'(C, c_0, c_s, \text{succ}(n)), \text{pr}_2 \Phi'(C, c_0, c_s, \text{succ}(n)))) \\ &=_C (\text{succ}(\text{pr}_1(\text{succ}(n), c_s(n, \text{pr}_2 \Phi'(C, c_0, c_s, n)))), \\ &\quad c_s(\text{pr}_1(\text{succ}(n), c_s(n, \Phi'(C, c_0, c_s, n))), \text{pr}_2 \Phi'(C, c_0, c_s, \text{succ}(n)))) \\ &=_C (\text{succ}(\text{succ}(n)), c_s(\text{succ}(n), \text{pr}_2 \Phi'(C, c_0, c_s, \text{succ}(n))))\end{aligned}$$

Where the step introducing the propositional equality is an application of the indiscernability of identicals as applied to the induction hypothesis. We have thus shown that  $P(n)$  holds for all  $n$ .<sup>1</sup> Applying  $\text{pr}_2$  to either side gives

$$\Phi(C, c_0, c_s, \text{succ}(n)) \equiv \text{pr}_2 \Phi(C, c_0, c_s, \text{succ}(n)) =_C \text{pr}_2(n, c_s(n, \Phi(C, c_0, c_s, n))) \equiv \text{pr}_2(n, c_s(n, \Phi(C, c_0, c_s, n)))$$

for all  $n$ , meaning that the defining equations hold propositionally. I need to learn more Coq to do this proof in that.

**Exercise 1.5 (p. 56)** Show that if we define  $A + B \equiv \sum_{(x:2)} \text{rec}_2(\mathcal{U}, A, B, x)$ , then we can give a definition of  $\text{ind}_{A+B}$  for which the definitional equalities stated in \S1.7 hold.

**Solution** Define  $A + B$  as stated. We need to define a function of type

$$\text{ind}'_{A+B} : \prod_{C:(A+B) \rightarrow \mathcal{U}} \left( \prod_{(a:A)} C(\text{inl}(a)) \right) \rightarrow \left( \prod_{(b:B)} C(\text{inr}(b)) \right) \rightarrow \prod_{(x:A+B)} C(x)$$

which means that we also need to define  $\text{inl}' : A \rightarrow A + B$  and  $\text{inr}' : B \rightarrow A + B$ ; these are

$$\text{inl}'(a) \equiv (0_2, a) \quad \text{inr}'(b) \equiv (1_2, b)$$

In Coq, we can use  $\text{sigT}$  to define  $\text{coprd}$  as a  $\Sigma$ -type: Suppose that  $C : A + B \rightarrow \mathcal{U}$ ,  $g_0 : \prod_{(a:A)} C(\text{inl}'(a))$ ,  $g_1 : \prod_{(b:B)} C(\text{inr}'(b))$ , and  $x : A + B$ ; we're looking to define

$$\text{ind}'_{A+B}(C, g_0, g_1, x)$$

We will use  $\text{ind}_{\sum_{(x:2)} \text{rec}_2(\mathcal{U}, A, B, x)}$ , and for notational convenience will write  $\Phi \equiv \sum_{(x:2)} \text{rec}_2(\mathcal{U}, A, B, x)$ .  $\text{ind}_\Phi$  has signature

$$\text{ind}_\Phi : \prod_{C:(\Phi) \rightarrow \mathcal{U}} \left( \prod_{(x:2)} \prod_{(y:\text{rec}_2(\mathcal{U}, A, B, x))} C((x, y)) \right) \rightarrow \prod_{(p:\Phi)} C(p)$$

So

$$\text{ind}_\Phi(C) : \left( \prod_{(x:2)} \prod_{(y:\text{rec}_2(\mathcal{U}, A, B, x))} C((x, y)) \right) \rightarrow \prod_{(p:\Phi)} C(p)$$

---

<sup>1</sup>Rather more sketchily than before I lost the first file—redo this?

To obtain something of type  $\prod_{(x:2)} \prod_{(y:\text{rec}_2(\mathcal{U}, A, B, x))} C((x, y))$  we'll have to use  $\text{ind}_2$ . In particular, for  $B(x) := \prod_{(y:\text{rec}_2(\mathcal{U}, A, B, x))} C((x, y))$  we have

$$\text{ind}_2(B) : B(0_2) \rightarrow B(1_2) \rightarrow \prod_{x:2} B(x)$$

along with

$$g_0 : \prod_{a:A} C(\text{inl}'(a)) \equiv \prod_{a:\text{rec}_2(\mathcal{U}, A, B, 0_2)} C((0_2, a)) \equiv B(0_2)$$

and similarly for  $g_1$ . So

$$\text{ind}_2(B, g_0, g_1) : \prod_{(x:2)} \prod_{(y:\text{rec}_2(\mathcal{U}, A, B, x))} C((x, y))$$

which is just what we needed for  $\text{ind}_\Phi$ . So we define

$$\text{ind}'_{A+B}(C, g_0, g_1, x) := \text{ind}_{\sum_{(x:2)} \text{rec}_2(\mathcal{U}, A, B, x)} \left( C, \text{ind}_2 \left( \prod_{y:\text{rec}_2(\mathcal{U}, A, B, x)} C((x, y)), g_0, g_1 \right), x \right)$$

and, in Coq, we use *sigT\_rect*, which is the built-in  $\lambda A. \lambda B. \text{ind}_{\sum_{(x:A)} B(x)}$ :

Now we must show that the definitional equalities

$$\text{ind}'_{A+B}(C, g_0, g_1, \text{inl}'(a)) \equiv g_0(a)$$

$$\text{ind}'_{A+B}(C, g_0, g_1, \text{inr}'(b)) \equiv g_1(b)$$

hold. For the first, we have

$$\begin{aligned} \text{ind}'_{A+B}(C, g_0, g_1, \text{inl}'(a)) &\equiv \text{ind}'_{A+B}(C, g_0, g_1, (0_2, a)) \\ &\equiv \text{ind}_{\sum_{(x:2)} \text{rec}_2(\mathcal{U}, A, B, x)} \left( C, \text{ind}_2 \left( \prod_{y:\text{rec}_2(\mathcal{U}, A, B, x)} C((x, y)), g_0, g_1 \right), (0_2, a) \right) \\ &\equiv \text{ind}_2 \left( \prod_{y:\text{rec}_2(\mathcal{U}, A, B, x)} C((x, y)), g_0, g_1, 0_2 \right) (a) \\ &\equiv g_0(a) \end{aligned}$$

and for the second,

$$\begin{aligned} \text{ind}'_{A+B}(C, g_0, g_1, \text{inr}'(b)) &\equiv \text{ind}'_{A+B}(C, g_0, g_1, (1_2, b)) \\ &\equiv \text{ind}_{\sum_{(x:2)} \text{rec}_2(\mathcal{U}, A, B, x)} \left( C, \text{ind}_2 \left( \prod_{y:\text{rec}_2(\mathcal{U}, A, B, x)} C((x, y)), g_0, g_1 \right), (1_2, b) \right) \\ &\equiv \text{ind}_2 \left( \prod_{y:\text{rec}_2(\mathcal{U}, A, B, x)} C((x, y)), g_0, g_1, 1_2 \right) (b) \\ &\equiv g_1(b) \end{aligned}$$

Trivial calculations, as Coq can attest:

**Exercise 1.6 (p. 56)** Show that if we define  $A \times B := \prod_{(x:2)} \text{rec}_2(\mathcal{U}, A, B, x)$ , then we can give a definition of  $\text{ind}_{A \times B}$  for which the definitional equalities stated in §1.5 hold propositionally (i.e. using equality types).

**Solution** Define

$$A \times B := \prod_{x:2} \text{rec}_2(\mathcal{U}, A, B, x)$$

Supposing that  $a : A$  and  $b : B$ , we have an element  $(a, b) : A \times B$  given by

$$(a, b) := \text{ind}_2(\text{rec}_2(\mathcal{U}, A, B), a, b)$$

Defining this type and constructor in Coq, we have

An induction principle for  $A \times B$  will, given a family  $C : A \times B \rightarrow \mathcal{U}$  and a function

$$g : \prod_{(x:A)} \prod_{(y:B)} C((x, y)),$$

give a function  $f : \prod_{(x:A \times B)} C(x)$  defined by

$$f((x, y)) := g(x)(y)$$

So suppose that we have such a  $C$  and  $g$ . Writing things out in terms of the definitions, we have

$$\begin{aligned} C : \left( \prod_{x:2} \text{rec}_2(\mathcal{U}, A, B, x) \right) &\rightarrow \mathcal{U} \\ g : \prod_{(x:A)} \prod_{(y:B)} C(\text{ind}_2(\text{rec}_2(\mathcal{U}, A, B), x, y)) \end{aligned}$$

We can define projections by

$$\text{pr}_1 p := p(0_2) \quad \text{pr}_2 p := p(1_2)$$

Since  $p$  is an element of a dependent type, we have

$$\begin{aligned} p(0_2) : \text{rec}_2(\mathcal{U}, A, B, 0_2) &\equiv A \\ p(1_2) : \text{rec}_2(\mathcal{U}, A, B, 1_2) &\equiv B \end{aligned}$$

which checks out. Then we have

$$\begin{aligned} g(\text{pr}_1 p)(\text{pr}_2 p) &: C(\text{ind}_2(\text{rec}_2(\mathcal{U}, A, B), (\text{pr}_1 p), (\text{pr}_2 p))) \\ &\equiv C(\text{ind}_2(\text{rec}_2(\mathcal{U}, A, B), (\text{pr}_1 p), (\text{pr}_2 p))) \\ &\equiv C((p(0_2), p(1_2))) \end{aligned}$$

So we have defined a function

$$f' : \prod_{p:A \times B} C((p(0_2), p(1_2)))$$

But we need one of the type

$$f : \prod_{p:A \times B} C(p)$$

To solve this problem, we need to appeal to function extensionality from \S2.9. This implies that there is a function

$$\text{funext} : \prod_{f,g:A \times B} \left( \prod_{x:2} (f(x) =_{\text{rec}_2(\mathcal{U}, A, B, x)} g(x)) \right) \rightarrow (f =_{A \times B} g)$$



So, consider

$$\text{funext}(p, (\text{pr}_1 p, \text{pr}_2 p)) : \left( \prod_{x:2} (p(x) =_{\text{rec}_2(\mathcal{U}, A, B, x)} (p(0_2), p(1_2))(x)) \right) \rightarrow (p =_{A \times B} (p(0_2), p(1_2)))$$

We just need to show that the antecedent is inhabited, which we can do with  $\text{ind}_2$ . So consider the family

$$\begin{aligned} E &:= \lambda(x:2). (p(x) =_{\text{rec}_2(\mathcal{U}, A, B, x)} (p(0_2), p(1_2))(x)) \\ &\equiv \lambda(x:2). (p(x) =_{\text{rec}_2(\mathcal{U}, A, B, x)} \text{ind}_2(\text{rec}_2(\mathcal{U}, A, B), p(0_2), p(1_2), x)) \end{aligned}$$

We have

$$\begin{aligned} E(0_2) &\equiv (p(0_2) =_{\text{rec}_2(\mathcal{U}, A, B, 0_2)} \text{ind}_2(\text{rec}_2(\mathcal{U}, A, B), p(0_2), p(1_2), 0_2)) \\ &\equiv (p(0_2) =_{\text{rec}_2(\mathcal{U}, A, B, 0_2)} p(0_2)) \end{aligned}$$

Thus  $\text{refl}_{p(0_2)} : E(0_2)$ . The same argument goes through to show that  $\text{refl}_{p(1_2)} : E(1_2)$ . This means that

$$h := \text{ind}_2(E, \text{refl}_{p(0_2)}, \text{refl}_{p(1_2)}) : \prod_{x:2} (p(x) =_{\text{rec}_2(\mathcal{U}, A, B, x)} (p(0_2), p(1_2)))$$

and thus

$$\text{funext}(p, (\text{pr}_1 p, \text{pr}_2 p), h) : p =_{A \times B} (p(0_2), p(1_2))$$

So, by the transport principle, there is a function

$$(\text{funext}(p, (\text{pr}_1 p, \text{pr}_2 p), h))_* : C((\text{pr}_1 p, \text{pr}_2 p)) \rightarrow C(p)$$

and we may define

$$\text{ind}_{A \times B}(C, g, p) := (\text{funext}(p, (\text{pr}_1 p, \text{pr}_2 p), h))_*(g(\text{pr}_1 p)(\text{pr}_2 p))$$

In Coq we can repeat this construction using *Funext*.

Now, we must show that the definitional equality holds propositionally. That is, we must show that the type

$$\text{ind}_{A \times B}(C, g, (a, b)) =_{C((a, b))} g(a)(b)$$

is inhabited. Unfolding the left hand side gives

$$\begin{aligned} \text{ind}_{A \times B}(C, g, (a, b)) &\equiv (\text{funext}((a, b), (\text{pr}_1(a, b), \text{pr}_2(a, b)), h))_*(g(\text{pr}_1(a, b))(\text{pr}_2(a, b))) \\ &\equiv (\text{funext}((a, b), (a, b), h))_*(g(a)(b)) \\ &\equiv \text{ind}_{=_{A \times B}}(D, d, (a, b), (a, b), \text{funext}((a, b), (a, b), h))(g(a)(b)) \end{aligned}$$

where  $D(x, y, \text{funext}((a, b), (a, b), h)) := C(x) \rightarrow C(y)$  and

$$d := \lambda x. \text{id}_{C(x)} : \prod_{x:A \times B} D(x, x, \text{refl}_x)$$

But the defining equality of

**Exercise 1.7 (p. 56)** Give an alternative derivation of  $\text{ind}'_{=A}$  from  $\text{ind}_{=A}$  which avoids the use of universes.

**Exercise 1.8 (p. 56)** Define multiplication and exponentiation using  $\text{rec}_{\mathbb{N}}$ . Verify that  $(\mathbb{N}, +, 0, \times, 1)$  is a semiring using only  $\text{ind}_{\mathbb{N}}$ .

**Solution** For multiplication, we need to construct a function  $\text{mult} : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ . Defined with pattern-matching, we would have

$$\begin{aligned}\text{mult}(0, m) &::= 0 \\ \text{mult}(\text{succ}(n), m) &::= m + \text{mult}(n, m)\end{aligned}$$

so in terms of  $\text{rec}_{\mathbb{N}}$  we have

$$\text{mult} ::= \text{rec}_{\mathbb{N}}(\mathbb{N} \rightarrow \mathbb{N}, \lambda n. 0, \lambda n. \lambda g. \lambda m. \text{add}(m, g(m)))$$

For exponentiation, we have the function  $\text{exp} : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ , with the intention that  $\text{exp}(e, b) = b^e$ . In terms of pattern matching,

$$\begin{aligned}\text{exp}(0, b) &::= 1 \\ \text{exp}(\text{succ}(e), b) &::= \text{mult}(b, \text{exp}(e, b))\end{aligned}$$

or, in terms of  $\text{rec}_{\mathbb{N}}$ ,

$$\text{exp} ::= \text{rec}_{\mathbb{N}}(\mathbb{N} \rightarrow \mathbb{N}, \lambda n. 1, \lambda n. \lambda g. \lambda m. \text{mult}(m, g(m)))$$

In Coq, we can define these by

To verify that  $(\mathbb{N}, +, 0, \times, 1)$  is a semiring, we need stuff from Chapter 2.

**Exercise 1.9 (p. 56)** Define the type family  $\text{Fin} : \mathbb{N} \rightarrow \mathcal{U}$  mentioned at the end of \S1.3, and the dependent function  $\text{fmax} : \prod_{(n:\mathbb{N})} \text{Fin}(n+1)$  mentioned in \S1.4.

**Solution**  $\text{Fin}(n)$  is a type with exactly  $n$  elements. Essentially, we want to recreate  $\mathbb{N}$  using types; so we will replace 0 with **0** and  $\text{succ}$  with a coproduct. So we define  $\text{Fin}$  recursively:

$$\begin{aligned}\text{Fin}(0) &::= \mathbf{0} \\ \text{Fin}(\text{succ}(n)) &::= \text{Fin}(n) + \mathbf{1}\end{aligned}$$

or, equivalently,

$$\text{Fin} ::= \text{rec}_{\mathbb{N}}(\mathcal{U}, \mathbf{0}, \lambda C. C + \mathbf{1})$$

**Exercise 1.10 (p. 56)** Show that the Ackermann function  $\text{ack} : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ , satisfying the following equations

$$\begin{aligned}\text{ack}(0, n) &\equiv \text{succ}(n), \\ \text{ack}(\text{succ}(m), 0) &\equiv \text{ack}(m, 1), \\ \text{ack}(\text{succ}(m), \text{succ}(n)) &\equiv \text{ack}(m, \text{ack}(\text{succ}(m), n)),\end{aligned}$$

is definable using only  $\text{rec}_{\mathbb{N}}$ .

**Solution** Define

$$\text{ack} ::= \text{rec}_{\mathbb{N}}(\mathbb{N} \rightarrow \mathbb{N}, \text{succ}, \lambda m. \lambda r. \text{rec}_{\mathbb{N}}(\mathbb{N}, r(1), \lambda n. \lambda s. r(s(r, n))))$$

To show that the defining equalities hold, we'll suppress the first argument of  $\text{rec}_{\mathbb{N}}$  for clarity. For the first we have

$$\text{ack}(0, n) \equiv \text{rec}_{\mathbb{N}}(\text{succ}, \lambda m. \lambda r. \text{rec}_{\mathbb{N}}(r(1), \lambda n. \lambda s. r(s(r, n))), 0)(n) \equiv \text{succ}(n)$$

For the second,

$$\begin{aligned}
& \text{ack}(\text{succ}(m), 0) \\
& \equiv \text{rec}_{\mathbb{N}}(\text{succ}, \lambda m. \lambda r. \text{rec}_{\mathbb{N}}(r(1), \lambda n. \lambda s. r(s(r, n))), \text{succ}(m))(0) \\
& \equiv ((\lambda r. \text{rec}_{\mathbb{N}}(r(1), \lambda n. \lambda s. r(s(r, n)))) \text{rec}_{\mathbb{N}}(\text{succ}, \lambda m. \lambda r. \text{rec}_{\mathbb{N}}(r(1), \lambda n. \lambda s. r(s(r, n))), m))(0) \\
& \equiv ((\lambda r. \text{rec}_{\mathbb{N}}(r(1), \lambda n. \lambda s. r(s(r, n)))) \text{ack}(m, -))(0) \\
& \equiv \text{rec}_{\mathbb{N}}(\text{ack}(m, 1), \lambda n. \lambda s. \text{ack}(m, s(\text{ack}(m, -), n)), 0) \\
& \equiv \text{ack}(m, 1)
\end{aligned}$$

Finally, using the first few steps of this second calculation again,

$$\begin{aligned}
& \text{ack}(\text{succ}(m), \text{succ}(n)) \\
& \equiv \text{rec}_{\mathbb{N}}(\text{succ}, \lambda m. \lambda r. \text{rec}_{\mathbb{N}}(r(1), \lambda n. \lambda s. r(s(r, n))), \text{succ}(m))(\text{succ}(n)) \\
& \equiv \text{rec}_{\mathbb{N}}(\text{ack}(m, 1), \lambda n. \lambda s. \text{ack}(m, s(\text{ack}(m, -), n)), \text{succ}(n)) \\
& \equiv (\lambda s. \text{ack}(m, s(\text{ack}(m, -), n))) \text{rec}_{\mathbb{N}}(\text{ack}(m, 1), \lambda n. \lambda s. \text{ack}(m, s(\text{ack}(m, -), n)), n)
\end{aligned}$$

**Exercise 1.11 (p. 56)** Show that for any type  $A$ , we have  $\neg\neg\neg A \rightarrow \neg A$ .

**Solution** Suppose that  $\neg\neg\neg A$  and  $A$ . Supposing further that  $\neg A$ , we get a contradiction with the second assumption, so  $\neg\neg A$ . But this contradicts the first assumption that  $\neg\neg\neg A$ , so  $\neg A$ . Discharging the first assumption gives  $\neg\neg\neg A \rightarrow \neg A$ .

In type-theoretic terms, the first assumption is  $x : ((A \rightarrow 0) \rightarrow 0) \rightarrow 0$ , and the second is  $a : A$ . If we further assume that  $h : A \rightarrow 0$ , then  $h(a) : 0$ , so discharging the  $h$  gives

$$\lambda(h : A \rightarrow 0). h(a) : (A \rightarrow 0) \rightarrow 0$$

But then we have

$$x(\lambda(h : A \rightarrow 0). h(a)) : 0$$

so discharging the  $a$  gives

$$\lambda(a : A). x(\lambda(h : A \rightarrow 0). h(a)) : A \rightarrow 0$$

And discharging the first assumption gives

$$\lambda(x : ((A \rightarrow 0) \rightarrow 0) \rightarrow 0). \lambda(a : A). x(\lambda(h : A \rightarrow 0). h(a)) : (((A \rightarrow 0) \rightarrow 0) \rightarrow 0) \rightarrow (A \rightarrow 0)$$

This is automatic for Coq, though not trivial. One nice thing is that we can get a proof out of Coq by printing this `Goal`. It returns

```

fun (A : Type) (X : ¬ ¬ ¬ A) (X0 : A) ⇒ X (fun X1 : A → Empty ⇒ X1 X0)
: ∀ A : Type, ¬ ¬ ¬ A → ¬ A

```

**Exercise 1.12 (p. 56)** Using the propositions as types interpretation, derive the following tautologies.

- (i) If  $A$ , then (if  $B$  then  $A$ ).
- (ii) If  $A$ , then not (not  $A$ ).
- (iii) If (not  $A$  or not  $B$ ), then not ( $A$  and  $B$ ).

**Solution** (i) Suppose that  $A$  and  $B$ ; then  $A$ . Discharging the assumptions,  $A \rightarrow B \rightarrow A$ . That is, we have

$$\lambda(a : A). \lambda(b : B). a : A \rightarrow B \rightarrow A$$

and in Coq,

(ii) Suppose that  $A$ . Supposing further that  $\neg A$  gives a contradiction, so  $\neg \neg A$ . That is,

$$\lambda(a : A). \lambda(f : A \rightarrow \mathbf{0}). f(a) : A \rightarrow (A \rightarrow \mathbf{0}) \rightarrow \mathbf{0}$$

(iii) Finally, suppose  $\neg A \vee \neg B$ . Supposing further that  $A \wedge B$  means that  $A$  and that  $B$ . There are two cases. If  $\neg A$ , then we have a contradiction; but also if  $\neg B$  we have a contradiction. Thus  $\neg(A \wedge B)$ .

Type-theoretically, we assume that  $x : (A \rightarrow \mathbf{0}) + (B \rightarrow \mathbf{0})$  and  $z : A \times B$ . Conjunction elimination gives  $\text{pr}_1 z : A$  and  $\text{pr}_2 z : B$ . We can now perform a case analysis. Suppose that  $x_A : A \rightarrow \mathbf{0}$ ; then  $x_A(\text{pr}_1 z) : \mathbf{0}$ , a contradiction; if instead  $x_B : B \rightarrow \mathbf{0}$ , then  $x_B(\text{pr}_2 z) : \mathbf{0}$ . By the recursion principle for the coproduct, then,

$$f(z) \equiv \text{rec}_{(A \rightarrow \mathbf{0}) + (B \rightarrow \mathbf{0})}(\mathbf{0}, \lambda x. x(\text{pr}_1 z), \lambda x. x(\text{pr}_2 z)) : (A \rightarrow \mathbf{0}) + (B \rightarrow \mathbf{0}) \rightarrow \mathbf{0}$$

Discharging the assumption that  $A \times B$  is inhabited, we have

$$f : A \times B \rightarrow (A \rightarrow \mathbf{0}) + (B \rightarrow \mathbf{0}) \rightarrow \mathbf{0}$$

So

$$\text{swap}(A \times B, (A \rightarrow \mathbf{0}) + (B \rightarrow \mathbf{0}), \mathbf{0}, f) : (A \rightarrow \mathbf{0}) + (B \rightarrow \mathbf{0}) \rightarrow A \times B \rightarrow \mathbf{0}$$

**Exercise 1.13 (p. 57)** Using propositions-as-types, derive the double negation of the principle of excluded middle, i.e.  $\sim \text{prove } \text{not } (\text{not } (P \text{ or } \text{not } P))$ .

**Solution** Suppose that  $\neg(P \vee \neg P)$ . Then, assuming  $P$ , we have  $P \vee \neg P$  by disjunction introduction, a contradiction. Hence  $\neg P$ . But disjunction introduction on this again gives  $P \vee \neg P$ , a contradiction. So we must reject the remaining assumption, giving  $\neg \neg(P \vee \neg P)$ .

In type-theoretic terms, the initial assumption is that  $g : P + (P \rightarrow \mathbf{0}) \rightarrow \mathbf{0}$ . Assuming  $p : P$ , disjunction introduction results in  $\text{inl}(p) : P + (P \rightarrow \mathbf{0})$ . But then  $g(\text{inl}(p)) : \mathbf{0}$ , so we discharge the assumption of  $p : P$  to get

$$\lambda(p : P). g(\text{inl}(p)) : P \rightarrow \mathbf{0}$$

Applying disjunction introduction again leads to contradiction, as

$$g(\text{inr}(\lambda(p : P). g(\text{inl}(p)))) : \mathbf{0}$$

So we must reject the assumption of  $\neg(P \vee \neg P)$ , giving the result:

$$\lambda(g : P + (P \rightarrow \mathbf{0}) \rightarrow \mathbf{0}). g(\text{inr}(\lambda(p : P). g(\text{inl}(p)))) : (P + (P \rightarrow \mathbf{0}) \rightarrow \mathbf{0}) \rightarrow \mathbf{0}$$

Finally, in Coq,

**Exercise 1.14 (p. 57)** Why do the induction principles for identity types not allow us to construct a function  $f : \prod_{(x:A)} \prod_{(p:x=x)} (p = \text{refl}_x)$  with the defining equation

$$f(x, \text{refl}_x) \equiv \text{refl}_{\text{refl}_x} \quad ?$$

**Exercise 1.15 (p. 57)** Show that indiscernability of identicals follows from path induction.