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 $f : A \to B$ and $g : B \to C$, define their **composite** $g \circ f : A \to C$. Show that we have $h \circ (g \circ f) \equiv (h \circ g) \circ f$.

Solution Define $g \circ f :\equiv \lambda(x : A).\, g(f(x))$. Then if $h : C \to D$, 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:Type} (g : B → C) (f : A → B) := fun x ⇒ g (f x).
Theorem compose_assoc : ∀ (A B C D : Type) (f : A → B) (g : B→ C) (h : C → D),
    compose h (compose g f) = compose (compose h g) f.
```

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

```
Section Exercise2a.
    Context {A B : Type}.
```

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

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

in terms of these projections as follows

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

or, in Coq,

```
Definition recprod (C : Type) (g : A → B → C) (p : A × B) := g (fst p) (snd p).
```

We must then show that

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

which in Coq is also trivial:

```
Goal ∀ C g a b, recprod C g (a, b) = g a b.
```

Now for the Σ-types. Here we have a projection

$$\mathsf{pr}_1 : \left(\sum_{x:A} B(x)\right) \to A$$

and another

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

Define a function of type

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

by

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

```
Definition recsm (C : Type) (g : ∀ (x : A), B x → C) (p : ∃ (x : A), B x) := g (projT1 p) (projT2 p).
```

We then verify that

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

which is again trivial in Coq:

```
Goal ∀ C g a b, recsm C g (a; b) = g a b.
```

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

Solution The induction principle has type

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

For a first pass, we can define

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

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

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

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

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

By the transport principle, there is a function

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

so

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

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

```
Definition uppt (x : A × B) : (fst x, snd x) = x.
Definition indprd (C : A × B → Type) (g : ∀ (x:A) (y:B), C (x, y)) (z : A × B) :=
    (uppt z) # (g (fst z) (snd z)).
```

We now have to show that

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

Unfolding the left gives

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

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

```
Goal ∀ C g a b, indprd C g (a, b) = g a b.
```

For Σ-types, we define

$$\mathsf{ind}_{\sum_{(x:A)} B(x)} : \prod_{C:(\sum_{(x:A)} B(x))\to\mathcal{U}} \left(\prod_{(a:A)}\prod_{(b:B(a))} C((a,b))\right) \to \prod_{p:\sum_{(x:A)} B(x)} C(p)$$

at first pass by

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

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

$$\mathsf{upst} : \prod_{p:\sum_{(x:A)} B(x)} \left((\mathsf{pr}_1 p, \mathsf{pr}_2 p) =_{\sum_{(x:A)} B(x)} p\right)$$

As for product types, we can define

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

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

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

and in Coq,

```
Definition upst (p : {x:A & B x}) : (projT1 p; projT2 p) = p.
Definition indsm (C : {x:A & B x} → Type) (g : ∀ (a:A) (b:B a), C (a; b)) (p : {x:A & B x}) :=
    (upst p) # (g (projT1 p) (projT2 p)).
```

Now we must verify that

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

We have

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

which Coq finds trivial:

```
Goal ∀ C g a b, indsm C g (a; b) = g a b.
```

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

$$\mathsf{iter} : \prod_{C:\mathcal{U}} C \to (C \to C) \to \mathbb{N} \to C$$

with the defining equations

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

derive a function having the type of the recursor $\mathsf{rec}_{\mathbb{N}}$. 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 $c_s : \mathbb{N} \to C \to C$. $\mathsf{iter}(C)$ allows for the n-fold application of a single function to a single input from C, whereas $\mathsf{rec}_{\mathbb{N}}$ 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

$$\mathsf{iter}_{\mathbb{N}\times C} : \mathbb{N} \times C \to (\mathbb{N} \times C \to \mathbb{N} \times C) \to \mathbb{N} \to \mathbb{N} \times C$$

to derive a function

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

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

The first argument of $\mathsf{iter}_{\mathbb{N}\times C}$ 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 succ to advance the second argument, representing the number of steps taken. This gives the function

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

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

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

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

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

which has the same type as $\mathsf{rec}_{\mathbb{N}}$. 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 Φ. To do this, we must show that

$$\Phi(C, c_0, c_s, 0) =_C c_0$$

$$\prod_{n:\mathbb{N}} \Big(\Phi_C(c_0, c_s, \mathsf{succ}(n)) =_C c_s(n, \Phi(C, c_0, c_s, n))\Big)$$

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

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

and in Coq,

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

So $\mathsf{refl}_{c_0} : \Phi(C, c_0, c_s, 0) =_C c_0$. This establishes the first equality. We prove the second by strengthening the induction hypothesis. Define Φ' as the argument of pr_2 in the above definition; i.e., such that $\Phi = \mathsf{pr}_2\Phi'$.

```
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 $n : \mathbb{N}$,

$$P(n) :\equiv \big(\Phi'(C, c_0, c_s, \mathsf{succ}(n)) =_C (n, c_s(n, \mathsf{pr}_2\Phi'(C, c_0, c_s, n)))\big).$$

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

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

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

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

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.[1] Applying pr_2 to either side gives

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

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 $A + B :\equiv \sum_{(x:\mathbf{2})} \mathsf{rec}_{\mathbf{2}}(\mathcal{U}, A, B, x)$, then we can give a definition of 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

$$\mathsf{ind}'_{A+B} : \prod_{C:(A+B)\to\mathcal{U}} \Big(\prod_{(a:A)} C(\mathsf{inl}(a))\Big) \to \Big(\prod_{(b:B)} C(\mathsf{inr}(b))\Big) \to \prod_{(x:A+B)} C(x)$$

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

$$\mathsf{inl}'(a) :\equiv (0_{\mathbf{2}}, a) \qquad\qquad \mathsf{inr}'(b) :\equiv (1_{\mathbf{2}}, b)$$

In Coq, we can use *sigT* to define coprd as a Σ-type:

```
Section Exercise5.

Context {A B : Type}.

Definition coprd := {x:Bool & if x then B else A}.
Definition myinl (a : A) := existT (fun x:Bool ⇒ if x then B else A) false a.
Definition myinr (b : B) := existT (fun x:Bool ⇒ if x then B else A) true b.
```

Suppose that $C : A + B \to \mathcal{U}$, $g_0 : \prod_{(a:A)} C(\mathsf{inl}'(a))$, $g_1 : \prod_{(b:B)} C(\mathsf{inr}'(b))$, and $x : A + B$; we're looking to define

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

[1] Rather more sketchily than before I lost the first file—redo this?

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

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

So

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

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

$$\mathsf{ind}_{\mathbf{2}}(B) : B(0_{\mathbf{2}}) \to B(1_{\mathbf{2}}) \to \prod_{x:\mathbf{2}} B(x)$$

along with

$$g_0 : \prod_{a:A} C(\mathsf{inl}'(a)) \equiv \prod_{a:\mathsf{rec}_{\mathbf{2}}(\mathcal{U},A,B,0_{\mathbf{2}})} C((0_{\mathbf{2}},a)) \equiv B(0_{\mathbf{2}})$$

and similarly for g_1. So

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

which is just what we needed for ind_Φ. So we define

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

and, in Coq, we use sigT_rect, which is the built-in $\mathsf{ind}_{\sum_{(x:A)} B(x)}$:

```
Definition indcoprd (C : coprd → Type) (g0 : ∀ a : A, C (myinl a)) (g1 : ∀ b : B, C (myinr b)) (x : coprd)
    :=
    sigT_rect C (Bool_rect (fun x:Bool ⇒ ∀ (y : if x then B else A), C (x; y)) g1 g0) x.
```

Now we must show that the definitional equalities

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

hold. For the first, we have

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

and for the second,

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

Trivial calculations, as Coq can attest:

```
Goal ∀ C g0 g1 a, indcoprd C g0 g1 (myinl a) = g0 a.
Goal ∀ C g0 g1 b, indcoprd C g0 g1 (myinr b) = g1 b.
```

Exercise 1.6 (p. 56) Show that if we define $A \times B :\equiv \prod_{(x:\mathbf{2})} \mathsf{rec}_{\mathbf{2}}(\mathcal{U}, A, B, x)$, then we can give a definition of $\mathsf{ind}_{A\times B}$ for which the definitional equalities stated in \S1.5 hold propositionally (i.e.~using equality types).

Solution Define

$$A \times B :\equiv \prod_{x:\mathbf{2}} \mathsf{rec}_{\mathbf{2}}(\mathcal{U}, A, B, x)$$

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

$$(a, b) :\equiv \mathsf{ind}_{\mathbf{2}}(\mathsf{rec}_{\mathbf{2}}(\mathcal{U}, A, B), a, b)$$

Defining this type and constructor in Coq, we have

```
Definition prd := ∀ x:Bool, if x then B else A.
Definition mypair (a:A) (b:B) := Bool_rect (fun x:Bool ⇒ if x then A else B) a b.
```

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

$$f((x, y)) :\equiv g(x)(y)$$

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

$$
\begin{aligned}
&C : \left(\prod_{x:\mathbf{2}} \mathsf{rec}_{\mathbf{2}}(\mathcal{U}, A, B, x)\right) \to \mathcal{U} \\
&g : \prod_{(x:A)} \prod_{(y:B)} C(\mathsf{ind}_{\mathbf{2}}(\mathsf{rec}_{\mathbf{2}}(\mathcal{U}, A, B), x, y))
\end{aligned}
$$

We can define projections by

$$\mathsf{pr}_1 p :\equiv p(0_{\mathbf{2}}) \qquad \mathsf{pr}_2 p :\equiv p(1_{\mathbf{2}})$$

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

$$
\begin{aligned}
&p(0_{\mathbf{2}}) : \mathsf{rec}_{\mathbf{2}}(\mathcal{U}, A, B, 0_{\mathbf{2}}) \equiv A \\
&p(1_{\mathbf{2}}) : \mathsf{rec}_{\mathbf{2}}(\mathcal{U}, A, B, 1_{\mathbf{2}}) \equiv B
\end{aligned}
$$

which checks out. Then we have

$$\begin{aligned} g(\mathsf{pr}_1 p)(\mathsf{pr}_2 p) &: C(\mathsf{ind}_{\mathbf{2}}(\mathsf{rec}_{\mathbf{2}}(\mathcal{U}, A, B), (\mathsf{pr}_1 p), (\mathsf{pr}_2 p))) \\ &\equiv C(\mathsf{ind}_{\mathbf{2}}(\mathsf{rec}_{\mathbf{2}}(\mathcal{U}, A, B), (\mathsf{pr}_1 p), (\mathsf{pr}_2 p))) \\ &\equiv C((p(0_{\mathbf{2}}), p(1_{\mathbf{2}}))) \end{aligned}$$

So we have defined a function

$$f' : \prod_{p:A\times B} C((p(0_{\mathbf{2}}), p(1_{\mathbf{2}})))$$

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

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

So, consider

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

We just need to show that the antecedent is inhabited, which we can do with $\mathsf{ind}_{\mathbf{2}}$. So consider the family

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

We have

$$\begin{aligned} E(0_{\mathbf{2}}) &\equiv (p(0_{\mathbf{2}}) =_{\mathsf{rec}_{\mathbf{2}}(\mathcal{U},A,B,0_{\mathbf{2}})} \mathsf{ind}_{\mathbf{2}}(\mathsf{rec}_{\mathbf{2}}(\mathcal{U}, A, B), p(0_{\mathbf{2}}), p(1_{\mathbf{2}}), 0_{\mathbf{2}})) \\ &\equiv (p(0_{\mathbf{2}}) =_{\mathsf{rec}_{\mathbf{2}}(\mathcal{U},A,B,0_{\mathbf{2}})} p(0_{\mathbf{2}})) \end{aligned}$$

Thus $\mathsf{refl}_{p(0_{\mathbf{2}})} : E(0_{\mathbf{2}})$. The same argument goes through to show that $\mathsf{refl}_{p(1_{\mathbf{2}})} : E(1_{\mathbf{2}})$. This means that

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

and thus

$$\mathsf{funext}(p, (\mathsf{pr}_1 p, \mathsf{pr}_2 p), h) : p =_{A\times B} (p(0_{\mathbf{2}}), p(1_{\mathbf{2}}))$$

So, by the transport principle, there is a function

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

and we may define

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

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

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

is inhabited. Unfolding the left hand side gives

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

where $D(x, y, \mathsf{funext}((a,b), (a,b), h)) :\equiv C(x) \to C(y)$ and

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

But the defining equality of

End Exercise6.

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

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

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

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

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

$$\mathsf{mult} :\equiv \mathsf{rec}_{\mathbb{N}}(\mathbb{N} \to \mathbb{N}, \lambda n.\, 0, \lambda n.\, \lambda g.\, \lambda m.\, \mathsf{add}(m, g(m)))$$

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

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

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

$$\mathsf{exp} :\equiv \mathsf{rec}_{\mathbb{N}}(\mathbb{N} \to \mathbb{N}, \lambda n.\, 1, \lambda n.\, \lambda g.\, \lambda m.\, \mathsf{mult}(m, g(m)))$$

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 $\mathsf{Fin} : \mathbb{N} \to \mathcal{U}$ mentioned at the end of \S1.3, and the dependent function $\mathsf{fmax} : \prod_{(n:\mathbb{N})} \mathsf{Fin}(n+1)$ mentioned in \S1.4.

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

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

or, equivalently,

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

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

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

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

Solution Define

$$\mathsf{ack} :\equiv \mathsf{rec}_{\mathbb{N}}\big(\mathbb{N} \to \mathbb{N}, \mathsf{succ}, \lambda m.\, \lambda r.\, \mathsf{rec}_{\mathbb{N}}\big(\mathbb{N}, r(1), \lambda n.\, \lambda s.\, r(s(r, n))\big)\big)$$

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

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

For the second,

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

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

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

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

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 $\neg\neg\neg A \to \neg A$.

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

$$\lambda(h : A \to \mathbf{0}).\, h(a) : (A \to \mathbf{0}) \to \mathbf{0}$$

But then we have

$$x(\lambda(h : A \to \mathbf{0}).\, h(a)) : \mathbf{0}$$

so discharging the a gives

$$\lambda(a : A).\, x(\lambda(h : A \to \mathbf{0}).\, h(a)) : A \to \mathbf{0}$$

And discharging the first assumption gives

$$\lambda(x:((A\to\mathbf{0})\to\mathbf{0})\to\mathbf{0}).\,\lambda(a:A).\,x(\lambda(h:A\to\mathbf{0}).\,h(a)):(((A\to\mathbf{0})\to\mathbf{0})\to\mathbf{0})\to(A\to\mathbf{0})$$

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, $A\to B\to A$. That is, we have

$$\lambda(a:A).\,\lambda(b:B).\,a:A\to B\to A$$

and in Coq,

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

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

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

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

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

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

So

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

Exercise 1.13 (p. 57) Using propositions-as-types, derive the double negation of the principle of excluded middle, i.e.˜prove *not (not (P or 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 $g:P+(P\to\mathbf{0})\to\mathbf{0}$. Assuming $p:P$, disjunction introduction results in $\mathsf{inl}(p):P+(P\to\mathbf{0})$. But then $g(\mathsf{inl}(p)):\mathbf{0}$, so we discharge the assumption of $p:P$ to get

$$\lambda(p:P).\,g(\mathsf{inl}(p)):P\to\mathbf{0}$$

Applying disjunction introduction again leads to contradiction, as

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

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

$$\lambda(g:P+(P\to\mathbf{0})\to\mathbf{0}).\,g(\mathsf{inr}(\lambda(p:P).\,g(\mathsf{inl}(p)))):(P+(P\to\mathbf{0})\to\mathbf{0})\to\mathbf{0}$$

Finally, in Coq,

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

$$f(x, \mathsf{refl}_x) :\equiv \mathsf{refl}_{\mathsf{refl}_x} \qquad ?$$

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