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

Proof: For $\mathcal{A}, \mathcal{B}, \mathcal{C} : \mathcal{U}$, define $\circ : (\mathcal{B} \to \mathcal{C}) \to (\mathcal{A} \to \mathcal{B}) \to \mathcal{A} \to \mathcal{C}$ as $\lambda g.\lambda f.\lambda x.g(f(x))$. Now, for each $x : \mathcal{A}$, one has $(h \circ (g \circ f))(x) \equiv h((g \circ f)(x)) \equiv h(g(f(x))) \equiv (h \circ g)(f(x)) \equiv ((h \circ g) \circ f)(x)$.

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

Proof: It suffices to show this for Σ-types as they generalize product types: define $\text{rec}_{\Sigma_{(x:\mathcal{A})}\mathcal{B}(x)}(\mathcal{C}, g, p) :\equiv g(\text{pr}_1\ p)(\text{pr}_2\ p)$ where $\mathcal{C} : \mathcal{U}$, $g : \Pi_{(x:\mathcal{A})}\mathcal{B}(x) \to \mathcal{C}$ and $p : \Sigma_{(x:\mathcal{A})}\mathcal{B}(x)$. Then, $\text{rec}_{\Sigma_{(x:\mathcal{A})}\mathcal{B}(x)}(\mathcal{C}, g, (a, b)) :\equiv g(\text{pr}_1\ (a, b))(\text{pr}_2\ (a, b)) \equiv g(a)(b)$.

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

Proof: Note that in the previous exercise, the uniqueness principle was not necessary because the type $\mathcal{C} : \mathcal{U}$ was not a function. Now, however, $\mathcal{C}$ must take inputs both of the form $(a, b) : \Sigma_{(x:\mathcal{A})}\mathcal{B}(x)$ *and* $x : \Sigma_{(x:\mathcal{A})}\mathcal{B}(x)$. One needs the uniqueness principle to associate one form to the other.

To reiterate, one needs to derive

$$\text{ind}_{\Sigma_{(x:\mathcal{A})}\mathcal{B}(x)} : \prod_{\mathcal{C}:(\Sigma_{(x:\mathcal{A})}\mathcal{B}(x))\to\mathcal{U}} \Big(\prod_{a:\mathcal{A}}\prod_{b:\mathcal{B}(a)} \mathcal{C}((a,b))\Big) \to \prod_{\Sigma_{(x:\mathcal{A})}\mathcal{B}(x)} \mathcal{C}(x)$$

This is *not* as simple as defining $\text{ind}_{\Sigma_{(x:\mathcal{A})}\mathcal{B}(x)}(\mathcal{C}, g, x) :\equiv g(\text{pr}_1\ x)(\text{pr}_2\ x)$. The problem is this: for arbitrary $x : \Sigma_{(x:\mathcal{A})}\mathcal{B}(x)$, one does not immediately know whether $g(\text{pr}_1\ x)(\text{pr}_2\ x) : \mathcal{C}(x)$. The only guarantee one has is that $g(\text{pr}_1\ x)(\text{pr}_2\ x) : \mathcal{C}((\text{pr}_1\ x, \text{pr}_2\ x))$. Now, by the uniqueness principle of product types, one has

$$\text{uniq}_{\Sigma_{(x:\mathcal{A})}\mathcal{B}(x)}\ x : (\text{pr}_1\ x, \text{pr}_2\ x) =_{\Sigma_{(x:\mathcal{A})}\mathcal{B}(x)} x$$

where $\text{uniq}_{\Sigma_{(x:\mathcal{A})}\mathcal{B}(x)}\ (a, b) :\equiv \text{refl}_{(a,b)}$. Also, by the recursion principle of identity types (i.e. the indiscernability of identicals),

$$\text{rec}_{=_{\Sigma_{(x:\mathcal{A})}\mathcal{B}(x)}}(\mathcal{C}, (\text{pr}_1\ x, \text{pr}_2\ x), x, \text{uniq}_{\Sigma_{(x:\mathcal{A})}\mathcal{B}(x)}\ x) : \mathcal{C}((\text{pr}_1\ x, \text{pr}_2\ x)) \to \mathcal{C}(x)$$

where $\text{rec}_{=_{\Sigma_{(x:\mathcal{A})}\mathcal{B}(x)}}(\mathcal{C}, x, x, \text{refl}_x) :\equiv \text{id}_{\mathcal{C}(x)}$. Only now can one finally use this to get $\text{ind}_{\Sigma_{(x:\mathcal{A})}\mathcal{B}(x)}$; for convenience, denote the previous function as $(\text{uniq}_{\Sigma_{(x:\mathcal{A})}\mathcal{B}(x)}\ x)_*$:

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

In the case where $x \equiv (a, b)$, one can verify the correctness of this definition:

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

Exercise 1.4. Assuming as given only the *iterator* for natural numbers

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

with the defining equations

$$\begin{gathered}\text{iter}(\mathcal{C}, c_0, c_s, 0) :\equiv c_0 \\ \text{iter}(\mathcal{C}, c_0, c_s, \text{succ}(n)) :\equiv c_s(\text{iter}(\mathcal{C}, c_0, c_s, n)),\end{gathered}$$

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 $\mathbb{N}$.

Proof: The difference between iter and $\text{rec}_{\mathbb{N}}$ is that the "next step" function of $\text{rec}_{\mathbb{N}}$ takes an input from $\mathbb{N}$. During recursive calls, this input is the predecessor of the current iterative index of $\text{rec}_{\mathbb{N}}$. As iter does not automatically keep track of this number, one must manually do so to define a $\text{rec}_{\mathbb{N}}$ look-a-like with iter.

This can be achieved using products. Call iter on $\mathbb{N} \times \mathcal{C}$ instead of $\mathcal{C}$, tracking the predecessor on the left and the main calculation on the right. This is analogous to keeping track of the iteration count in "for-loop"s. More concretely, given a type $\mathcal{C} : \mathcal{U}$, an initial element $c_0 : \mathcal{C}$, a "next step" function $c_s : \mathbb{N} \to \mathcal{C} \to \mathcal{C}$ and the current iterative index $n : \mathbb{N}$, define

$$\begin{aligned}
c_0' &:\equiv (0, c_0) : \mathbb{N} \times \mathcal{C} \\
c_s' &:\equiv \lambda x.(\mathrm{succ}(\mathrm{pr}_1\ x), c_s\ x) : \mathbb{N} \times \mathcal{C} \to \mathbb{N} \times \mathcal{C} \\
\mathrm{rec}_{\mathbb{N}}'(\mathcal{C}, c_0, c_s, n) &:\equiv \mathrm{pr}_2\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', n) : \mathcal{C}
\end{aligned}$$

Note that for convenience, I write $c_s\ x$ instead of $c_s(\mathrm{pr}_1\ x, \mathrm{pr}_2\ x)$. One can verify this definition for a few values:

$$\begin{aligned}
\mathrm{rec}_{\mathbb{N}}'(\mathcal{C}, c_0, c_s, 0) &\equiv \mathrm{pr}_2\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', 0) \equiv \mathrm{pr}_2\ c_0' \equiv \mathrm{pr}_2\ (0, c_0) \equiv c_0 \\
\mathrm{rec}_{\mathbb{N}}'(\mathcal{C}, c_0, c_s, 1) &\equiv \mathrm{pr}_2\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', 1) \equiv \mathrm{pr}_2(c_s'\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', 0)) \\
&\equiv \mathrm{pr}_2((\lambda x.(\mathrm{succ}(\mathrm{pr}_1\ x), c_s\ x))\ (0, c_0)) \\
&\equiv \mathrm{pr}_2\ (1, c_s(0, c_0)) \equiv c_s(0, c_0) \\
\mathrm{rec}_{\mathbb{N}}'(\mathcal{C}, c_0, c_s, 2) &\equiv \mathrm{pr}_2\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', 2) \equiv \mathrm{pr}_2(c_s'\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', 1)) \\
&\equiv \mathrm{pr}_2((\lambda x.(\mathrm{succ}(\mathrm{pr}_1\ x), c_s\ x))\ (1, c_s(0, c_0))) \\
&\equiv \mathrm{pr}_2\ (2, c_s(1, c_s(0, c_0))) \equiv c_s(1, c_s(0, c_0))
\end{aligned}$$

which are indeed the expressions returned by using the usual $\mathrm{rec}_{\mathbb{N}}$ on the corresponding values. However, it is *not* immediately clear that this $\mathrm{rec}_{\mathbb{N}}'$ satisfies the judgmental equalities of $\mathrm{rec}_{\mathbb{N}}$. In particular, while the first equality is satisfied: $\mathrm{rec}_{\mathbb{N}}'(\mathcal{C}, c_0, c_s, 0) \equiv c_0$ as seen above, the second equality is problematic:

$$\begin{aligned}
\mathrm{rec}_{\mathbb{N}}'(\mathcal{C}, c_0, c_s, \mathrm{succ}(n)) &\equiv \mathrm{pr}_2\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', \mathrm{succ}(n)) \equiv \mathrm{pr}_2(c_s'\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', n)) \\
&\equiv \mathrm{pr}_2\ (\mathrm{succ}(\mathrm{pr}_1\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', n)), c_s\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', n)) \\
&\equiv c_s(\mathrm{pr}_1\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', n), \mathrm{pr}_2\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', n)) \\
&\equiv c_s(\mathrm{pr}_1\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', n), \mathrm{rec}_{\mathbb{N}}'(\mathcal{C}, c_0, c_s, n))
\end{aligned}$$

which is almost the same but nevertheless different from the usual $\mathrm{rec}_{\mathbb{N}}(\mathcal{C}, c_0, c_s, \mathrm{succ}(n)) \equiv c_s(n, \mathrm{rec}_{\mathbb{N}}(\mathcal{C}, c_0, c_s, n))$. In fact, it is impossible, without further type rules, to *judgmentally* equate $\mathrm{pr}_1\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', n)$ to n and, hence, to equate $\mathrm{rec}_{\mathbb{N}}'(\mathcal{C}, c_0, c_s, \mathrm{succ}(n))$ to $c_s(n, \mathrm{rec}_{\mathbb{N}}'(\mathcal{C}, c_0, c_s, n))$. However, one can get a weaker *propositional* version of the equality using induction. Before one gets to that though, one needs convenient names to deal with long unwieldy terms:

$$\begin{aligned}
\mathrm{recnum}_n &:\equiv \mathrm{pr}_1\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', n) \\
\hat{c}_s(x, n) &:\equiv c_s(x, \mathrm{rec}_{\mathbb{N}}'(\mathcal{C}, c_0, c_s, n))
\end{aligned}$$

To construct the required induction, one must first construct its inputs. The first two are easy:

$$\begin{aligned}
\mathcal{I} &:\equiv \lambda n.\mathrm{recnum}_n =_{\mathbb{N}} n : \mathbb{N} \to \mathcal{U} \\
i_0 &:\equiv \mathrm{refl}_0 : \mathcal{I}(0) \equiv \mathrm{recnum}_0 =_{\mathbb{N}} 0 \equiv 0 =_{\mathbb{N}} 0
\end{aligned}$$

The "next-step" input i_s is more involved. Given $n : \mathbb{N}$ and $p_n : \mathrm{recnum}_n =_{\mathbb{N}} n$, one must show $\mathrm{recnum}_{\mathrm{succ}(n)} =_{\mathbb{N}} \mathrm{succ}(n)$ i.e. $\mathrm{succ}(\mathrm{recnum}_n) =_{\mathbb{N}} \mathrm{succ}(n)$ (expand out $\mathrm{recnum}_{\mathrm{succ}(n)}$ to see this). This requires the indiscernability of identicals $\mathrm{rec}_{=_{\mathbb{N}}}$:

$$\begin{aligned}
\mathcal{E} &:\equiv \lambda x.\mathrm{succ}(\mathrm{recnum}_n) =_{\mathbb{N}} \mathrm{succ}(x) : \mathbb{N} \to \mathcal{U} \\
\mathrm{refl}_{\mathrm{succ}(\mathrm{recnum}_n)} &: \mathcal{E}(\mathrm{recnum}_n) \equiv \mathrm{succ}(\mathrm{recnum}_n) =_{\mathbb{N}} \mathrm{succ}(\mathrm{recnum}_n) \\
\mathrm{rec}_{=_{\mathbb{N}}}(\mathcal{E}, \mathrm{recnum}_n, n, p_n, \mathrm{refl}_{\mathrm{succ}(\mathrm{recnum}_n)}) &: \mathcal{E}(n) \equiv \mathrm{succ}(\mathrm{recnum}_n) =_{\mathbb{N}} \mathrm{succ}(n) \quad \textit{(the required inhabitant)}
\end{aligned}$$

Thus one can define

$$i_s :\equiv \lambda n.\lambda p_n.\mathrm{rec}_{=_{\mathbb{N}}}(\mathcal{E}, \mathrm{recnum}_n, n, p_n, \mathrm{refl}_{\mathrm{succ}(\mathrm{recnum}_n)}) : \prod_{(n:\mathbb{N})} \mathcal{I}(n) \to \mathcal{I}(\mathrm{succ}(n))$$

and one gets the sought-for propositional equality

$$\mathrm{ind}_{\mathbb{N}}(\mathcal{I}, i_0, i_s) : \prod_{(n:\mathbb{N})} \mathrm{recnum}_n =_{\mathbb{N}} n \equiv \prod_{(n:\mathbb{N})} (\mathrm{pr}_1\ \mathrm{iter}(\mathbb{N} \times \mathcal{C}, c_0', c_s', n)) =_{\mathbb{N}} n$$

Using this, apply the indiscernability of identicals again to finally prove $\mathrm{rec}_{\mathbb{N}}'(\mathcal{C}, c_0, c_s, \mathrm{succ}(n)) =_{\mathbb{N}} c_s(n, \mathrm{rec}_{\mathbb{N}}'(\mathcal{C}, c_0, c_s, n))$ with

$$\mathrm{rec}_{=_{\mathbb{N}}}(\lambda x.\hat{c}_s(\mathrm{recnum}_n, n) =_{\mathcal{C}} \hat{c}_s(x, n), \mathrm{recnum}_n, n, \mathrm{ind}_{\mathbb{N}}(\mathcal{I}, i_0, i_s, n), \mathrm{refl}_{\hat{c}_s(\mathrm{recnum}_n, n)})$$

Exercise 1.9. Define the type family $\mathrm{Fin} : \mathbb{N} \to \mathcal{U}$ mentioned at the end of §1.3, and the dependent function $\mathrm{fmax} : \prod_{(n:N)} \mathrm{Fin}(n+1)$ mentioned in §1.4.

Proof: With higher-order functions, this is easy. Assuming that the base universe is $\mathcal{U}_i$ and that there is a natural number recursor $\mathrm{rec}_{\mathbb{N},\mathcal{U}_{i+1}}$ in $\mathcal{U}_{i+1}$, define $\mathrm{Fin} :\equiv \mathrm{rec}_{\mathbb{N},\mathcal{U}_{i+1}}(\mathcal{U}_i, \mathbf{0}, \lambda n.\lambda \mathrm{Fin}_n.\mathrm{Fin}_n + \mathbf{1})$.

If the assumptions in the definition above are too much, one can define $\mathrm{Fin} : \mathbb{N} \to \mathcal{U}$ axiomatically in the same universe: assume, outright, a family $\mathrm{Fin} : \mathbb{N} \to \mathcal{U}$. To use this family, assume also $\mathrm{finzero} : \prod_{(n:\mathbb{N})} \mathrm{Fin}(\mathrm{succ}(n))$ and $\mathrm{finsucc} : \prod_{(n:\mathbb{N})} \mathrm{Fin}(n) \to \mathrm{Fin}(\mathrm{succ}(n))$. These are the constructors of Fin: they help one construct inhabitants of each $\mathrm{Fin}(n)$.

As an example of how Fin and its constructors are used, consider how one might construct inhabitants of $\mathrm{Fin}(0)$. Well, there is no way to do so with the given constructors. None of them help one find such an element. When finzero is evaluated at the lowest possible natural number 0, it merely produces an inhabitant $\mathrm{finzero}(0)$ in $\mathrm{Fin}(\mathrm{succ}(0)) \equiv \mathrm{Fin}(1)$. A similar story transpires with the other constructor finsucc. This situation corresponds with the idea that $\mathrm{Fin}(0)$ should be empty.

Now consider how one can find elements in $\mathrm{Fin}(1)$. As one saw before, $\mathrm{finzero}(0) : \mathrm{Fin}(1)$. And that is, in fact, it. If one tries to get another element through finsucc, note that $\mathrm{finsucc}(0) : \mathrm{Fin}(0) \to \mathrm{Fin}(1)$. If only one could get an element of $\mathrm{Fin}(0)$, one could use this function to get another inhabitant of $\mathrm{Fin}(1)$. But, as one saw above, there is no such way *using the constructors alone*. And hence, one can construct only one inhabitant of $\mathrm{Fin}(1)$. This corresponds with the idea that $\mathrm{Fin}(1)$ should have only one element 0_1.

Exercise 1.11. Show that for any type $\mathcal{A}$, we have $\neg\neg\neg\mathcal{A} \to \neg\mathcal{A}$.

Proof: In other words, one needs an inhabitant of $(((\mathcal{A} \to \mathbf{0}) \to \mathbf{0}) \to \mathbf{0}) \to \mathcal{A} \to \mathbf{0}$. Well, assume as given an inhabitant $x^{\neg\neg\neg} : ((\mathcal{A} \to \mathbf{0}) \to \mathbf{0}) \to \mathbf{0}$ and an inhabitant $x : \mathcal{A}$. One now needs to produce an inhabitant of $\mathbf{0}$.

Note that $x^{\neg\neg\neg}$ takes an input of $(\mathcal{A} \to \mathbf{0}) \to \mathbf{0}$ to produce an output of $\mathbf{0}$. So, the idea is to find an input of $(\mathcal{A} \to \mathbf{0}) \to \mathbf{0}$ and apply $x^{\neg\neg\neg}$ to it. But an input of $(\mathcal{A} \to \mathbf{0}) \to \mathbf{0}$ must itself take an input of $\mathcal{A} \to \mathbf{0}$ to produce an output of $\mathbf{0}$. This is easy to construct given an $x : \mathcal{A}$. For any $x^{\neg} : \mathcal{A} \to \mathbf{0}$, simply apply it to $x : \mathcal{A}$:- $x^{\neg}(x) : \mathbf{0}$. Given the previous, one can see that $x^{\neg\neg\neg}(\lambda x^{\neg}.x^{\neg}(x)) : \mathbf{0}$. Thus, the desired inhabitant is:

$$\lambda x^{\neg\neg\neg}.\lambda x.x^{\neg\neg\neg}(\lambda x^{\neg}.x^{\neg}(x)) : (((\mathcal{A} \to \mathbf{0}) \to \mathbf{0}) \to \mathbf{0}) \to \mathcal{A} \to \mathbf{0}$$

In fact, one can produce an inhabitant of the converse too:

$$\lambda x^{\neg}.\lambda x^{\neg\neg}.x^{\neg\neg}(x^{\neg}) : (\mathcal{A} \to \mathbf{0}) \to ((\mathcal{A} \to \mathbf{0}) \to \mathbf{0}) \to \mathbf{0}$$

Note: One *cannot* repeat the above steps to produce an inhabitant of $\neg\neg\mathcal{A} \to \mathcal{A}$. It is not clear how one can produce an $x : \mathcal{A}$ given a function $x^{\neg\neg} : (\mathcal{A} \to \mathbf{0}) \to \mathbf{0}$. For one thing, one does not have anything to apply this function to. For another thing, even if one could apply this function to something, the output of the function will be an inhabitant of $\mathbf{0}$ and not necessarily an inhabitant of $\mathcal{A}$. However, one can produce an inhabitant of its converse $\mathcal{A} \to \neg\neg\mathcal{A}$ as one did before:

$$\lambda x.\lambda x^{\neg}.x^{\neg}(x) : \mathcal{A} \to (\mathcal{A} \to \mathbf{0}) \to \mathbf{0} \equiv \mathcal{A} \to \neg\neg\mathcal{A}$$

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

Proof: In other words, one has to find an inhabitant of $((\mathcal{P} + (\mathcal{P} \to \mathbf{0})) \to \mathbf{0}) \to \mathbf{0}$. However, it is easier to prove $((\mathcal{P} \to \mathbf{0}) \times ((\mathcal{P} \to \mathbf{0}) \to \mathbf{0})) \to \mathbf{0}$ first.

To do so, given an inhabitant $\mathit{contra} : (\mathcal{P} \to \mathbf{0}) \times ((\mathcal{P} \to \mathbf{0}) \to \mathbf{0})$, one must find an inhabitant of $\mathbf{0}$. Note that the type on the product's left side, $\mathcal{P} \to \mathbf{0}$, is the same as the input of the type on the product's right side, $(\mathcal{P} \to \mathbf{0}) \to \mathbf{0}$. Thus, one can decompose *contra* to its components and apply the right component to the left one: $(\mathrm{pr}_2\ \mathit{contra})\ \mathrm{pr}_1\ \mathit{contra} : \mathbf{0}$.

Now, to complete the overall proof one needs to go from an inhabitant $\mathit{xmid}^{\neg} : (\mathcal{P} + (\mathcal{P} \to \mathbf{0})) \to \mathbf{0}$ to some inhabitant $(?, ?) : (\mathcal{P} \to \mathbf{0}) \times ((\mathcal{P} \to \mathbf{0}) \to \mathbf{0})$.

- To fill in the "left question mark", one needs some function $? : \mathcal{P} \to \mathbf{0}$ i.e. one needs to get some $? : \mathbf{0}$ given a $p : \mathcal{P}$. But if one has a $p : \mathcal{P}$, one also has $\mathrm{inl}\ p : \mathcal{P} + (\mathcal{P} \to \mathbf{0})$; so $\mathit{xmid}^{\neg}(\mathrm{inl}\ p) : \mathbf{0}$. Thus, $\lambda p.\ \mathit{xmid}^{\neg}(\mathrm{inl}\ p) : \mathcal{P} \to \mathbf{0}$.

- Similarly, to fill in the right question mark one has $\lambda p^{\neg}.\ \mathit{xmid}^{\neg}(\mathrm{inr}\ p^{\neg}) : (\mathcal{P} \to \mathbf{0}) \to \mathbf{0}$.

Combining all these together, one has the following proof of the original proposition:

$$\begin{aligned}
&\lambda\, xmid^{\neg}.(\mathrm{pr}_2\ (\lambda p.\ xmid^{\neg}(\mathrm{inl}\ p), \lambda p^{\neg}.\ xmid^{\neg}(\mathrm{inr}\ p^{\neg})))(\mathrm{pr}_1\ (\lambda p.\ xmid^{\neg}(\mathrm{inl}\ p), \lambda p^{\neg}.\ xmid^{\neg}(\mathrm{inr}\ p^{\neg})))\\
\equiv\ &\lambda\, xmid^{\neg}.(\lambda p^{\neg}.\ xmid^{\neg}(\mathrm{inr}\ p^{\neg}))(\lambda p.\ xmid^{\neg}(\mathrm{inl}\ p))\\
\equiv\ &\lambda\, xmid^{\neg}.\ xmid^{\neg}(\mathrm{inr}\ \lambda p.\ xmid^{\neg}(\mathrm{inl}\ p)) : ((\mathcal{P} + (\mathcal{P} \to \mathbf{0})) \to \mathbf{0}) \to \mathbf{0}
\end{aligned}$$