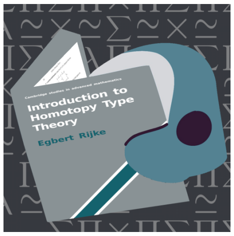

Worksheet 4 (Solved)

HoTTEST Summer School 2022

The HoTTEST TAs

11 July 2022

1 ($\star$)

We define the standard finite types $\mathsf{Fin} : \mathbb{N} \to \mathcal{U}_0$ inductively with constructors

$$\begin{aligned} \mathsf{pt} &: \Pi_{n:\mathbb{N}} \mathsf{Fin}(\mathsf{suc}(n)) \\ \mathsf{i} &: \Pi_{n:\mathbb{N}} \mathsf{Fin}(n) \to \mathsf{Fin}(\mathsf{suc}(n)). \end{aligned}$$

Spell out all elements of $\mathsf{Fin}(3)$.

The elements are

$$\begin{aligned} &\mathbf{pt}(2), \\ &\mathbf{i}(2, \mathbf{pt}(1)), \\ &\mathbf{i}(2, \mathbf{i}(1, \mathbf{pt}(0))). \end{aligned}$$

It is common practice to leave the argument n of the constructors implicit. Then the induction principle states that a dependent function

$$f : \Pi_{n:\mathbb{N}} \Pi_{x:\mathsf{Fin}(n)} P_n(x)$$

is determined by

$$g_n : \Pi_{x:\mathsf{Fin}(n)} P_n(x) \to P_{\mathsf{suc}(n)}(\mathsf{i}(x))$$

and

$$p_n : P_{\mathsf{suc}(n)}(\mathsf{pt}).$$

The function f satisfies the judgemental equalities

$$\begin{aligned} f_{\mathsf{suc}(n)}(\mathsf{i}(x)) &\doteq g_n(x, f_n(x)) \\ f_{\mathsf{suc}(n)}(\mathsf{pt}) &\doteq p_n. \end{aligned}$$

2 $(\star\star\star)$

It is also possible to define the standard finite types $\mathsf{Fin'} : \mathbb{N} \to \mathcal{U}_0$ recursively as a type family over $\mathbb{N}$,

$$\mathsf{Fin'}(0) \doteq \emptyset$$
$$\mathsf{Fin'}(\mathsf{suc}(n)) \doteq \mathsf{Fin'}(n) + \mathbb{1}.$$

We suggestively use the notation $\mathsf{i'} : \mathsf{Fin'}_n \to \mathsf{Fin'}_{\mathsf{suc}(n)}$ and $\mathsf{pt'} : \mathsf{Fin'}_{\mathsf{suc}(n)}$ for the inclusions inl and inr into the coproduct $\mathsf{Fin'}(n) + \mathbb{1}$. Formulate the induction principle of $\mathsf{Fin'}$.

The induction principle given to $\mathsf{Fin'}$ is exactly (the primed version of) the induction principle carried by Fin, described above.

3 $(\star\star)$

Choose your favourite version of the finite types. Use pattern matching to define two different inclusions $\iota, \hat{\iota} : \Pi_{n:\mathbb{N}}\mathsf{Fin}(n) \to \mathbb{N}$, such that the images of $\iota_{\mathsf{suc}(n)}$ and $\hat{\iota}_{\mathsf{suc}(n)}$ are the first $n+1$ natural numbers.

We define

$$\iota_{\mathsf{suc}(n)}(\mathsf{i}(x)) \doteq \iota_n(x)$$
$$\iota_{\mathsf{suc}(n)}(\mathsf{pt}) \doteq n$$
$$\hat{\iota}_{\mathsf{suc}(n)}(\mathsf{i}(x)) \doteq \mathsf{suc}(\iota'(x))$$
$$\hat{\iota}_{\mathsf{suc}(n)}(\mathsf{pt}) \doteq 0.$$

It is not necessary to define ι_0 because $\mathsf{Fin}(0)$ is empty.

4 $(\star)$

Give a recursive definition of the ordering relation $\leq : \mathbb{N} \to \mathbb{N} \to \mathcal{U}_0$.

Using induction on $\mathbb{N}$ twice we may define

$$0 \leq 0 \doteq \mathbb{1}$$
$$m +_{\mathbb{N}} 1 \leq 0 \doteq \emptyset$$
$$0 \leq n +_{\mathbb{N}} 1 \doteq \mathbb{1}$$
$$m +_{\mathbb{N}} 1 \leq n +_{\mathbb{N}} 1 \doteq m \leq n$$

5 $(\star\star)$

Define $\mathsf{is\text{-}prime} : \mathbb{N} \to \mathsf{Type}$.

There are various ways of defining this property. The one implemented in the repository is

$$\textbf{is-prime}(n) \doteq (2 \leq n) \times (\Pi_{x,y:\mathbb{N}}(x *_{\mathbb{N}} y = n) \to (x = 1) + (x = n)).$$

Egbert's book uses

$$\textbf{is-prime'}(n) \doteq \Pi_{d:\mathbb{N}}((d \neq n) \times (d \mid n)) \leftrightarrow (d = 1).$$

6 $(\star\star)$

State the twin prime conjecture and Goldbach's conjecture in HoTT.

The twin prime conjecture is

$$\Pi_{n:\mathbb{N}}\Sigma_{p:\mathbb{N}}((n \leq p) \times \textbf{is-prime}(p) \times \textbf{is-prime}(p +_{\mathbb{N}} 2)).$$

Goldbach's conjecture can be phrased

$$\Pi_{n:\mathbb{N}}\left(((4 \leq n) \times \textbf{is-even}(n)) \to \Sigma_{p,q:\mathbb{N}}(\textbf{is-prime}(p) \times \textbf{is-prime}(q) \times (n = p +_{\mathbb{N}} q))\right).$$

7 $(\star\star)$

Suppose we had constructed a proof

$$\mathsf{infinitude\text{-}of\text{-}primes} : \Pi_{n:\mathbb{N}}\Sigma_{p:\mathbb{N}}(\mathsf{is\text{-}prime}(p) \times (n < p)).$$

Further assume that the prime p returned by this program is the least prime above n. A definition of such a term can be found in the Agda UniMath library[1]. Construct a function $\mathsf{prime} : \mathbb{N} \to \mathbb{N}$ which computes the n-th prime.

We inductively define

$$\begin{aligned} \textbf{prime}(0) &\doteq 2 \\ \textbf{prime}(\textbf{suc}(n)) &\doteq \textbf{pr}_1(\textbf{infinitude-of-primes}(\textbf{prime}(n))). \end{aligned}$$

[1] `https://unimath.github.io/agda-unimath/elementary-number-theory.infinitude-of-primes.html`

8 $(\star\star)$

We define the predicate

$$\mathsf{is\text{-}decidable}(A) \doteq A + \neg A$$

for an arbitrary type A. Do we expect

$$\Pi_{n:\mathbb{N}}\mathsf{is\text{-}decidable}(\mathsf{is\text{-}prime}(n))$$

to be true (inhabited)? Why or why not?

We expect this to be true because it's easy to write down an algorithm which checks if a number is prime on paper. In fact, a proof in Agda is referenced on the same UniMath docs page.

9 $(\star\star\star)$

Suppose we had a proof

$$\mathsf{is\text{-}decidable\text{-}is\text{-}prime} : \Pi_{n:\mathbb{N}}\mathsf{is\text{-}decidable}(\mathsf{is\text{-}prime}(n)).$$

Construct a function

$$\mathsf{prime\text{-}counting} : \mathbb{N} \to \mathbb{N}$$

which computes the number of primes less than or equal to its input.

As usual, we define this function inductively. We put

$$\mathsf{prime\text{-}counting}(0) \doteq 0.$$

For the inductive step, $\mathsf{is\text{-}decidable\text{-}is\text{-}prime}$ allows us to proceed by case analysis on whether or not $n+1$ is a prime number. In other words, we may define

$$\begin{aligned} \mathsf{if\text{-}prime} &: \mathsf{is\text{-}decidable}(\mathsf{is\text{-}prime}(\mathsf{suc}(n))) \to \mathbb{N} \\ \mathsf{if\text{-}prime}(\mathsf{inl}(x)) &\doteq \mathsf{suc}(\mathsf{prime\text{-}counting}(n)) \\ \mathsf{if\text{-}prime}(\mathsf{inr}(x)) &\doteq \mathsf{prime\text{-}counting}(n) \end{aligned}$$

and put

$$\mathsf{prime\text{-}counting}(\mathsf{suc}(n)) \doteq \mathsf{if\text{-}prime}(\mathsf{is\text{-}decidable\text{-}is\text{-}prime}(\mathsf{suc}(n))).$$

10 $(\star\star\star)$

Show that adding k is an injective function which respects equality, i.e. that

$$(m = n) \leftrightarrow (m +_{\mathbb{N}} k = n +_{\mathbb{N}} k)$$

for all $m, n, k : \mathbb{N}$.

A proof of $(m = n) \to (m +_{\mathbb{N}} k = n +_{\mathbb{N}} k)$ is given by the action of the function

$$\lambda x.x +_{\mathbb{N}} k : \mathbb{N} \to \mathbb{N}$$

on paths $p : (m = n)$.
For the converse direction we induct on k. In the base case we need to show that $(m +_{\mathbb{N}} 0 = n +_{\mathbb{N}} 0) \to (m = n)$. Assume we have $p : m +_{\mathbb{N}} 0 = n +_{\mathbb{N}} 0$. By two applications of

$$\textsf{concat} : \Pi_{x,y,z:A}(x = y) \to ((y = z) \to (x = z)),$$

a sequence of identifications

$$m = (m +_{\mathbb{N}} 0) = (n +_{\mathbb{N}} 0) = n$$

implies $m = n$. The identification in the middle is proved by p. Since addition was defined by induction on the right argument, the outer identities hold judgementally. If $+$ had been defined by induction on the first argument, $m = m +_{\mathbb{N}} 0$ can be proved inductively.
In the inductive step we need to prove

$$((m +_{\mathbb{N}} \textsf{suc}(k)) = (n +_{\mathbb{N}} \textsf{suc}(k))) \to (m = n).$$

The induction hypothesis is of type

$$((m +_{\mathbb{N}} k) = (n +_{\mathbb{N}} k)) \to (m = n),$$

so by function composition it suffices to construct a proof of

$$((m +_{\mathbb{N}} \textsf{suc}(k)) = (n +_{\mathbb{N}} \textsf{suc}(k))) \to ((m +_{\mathbb{N}} k) = (n +_{\mathbb{N}} k)).$$

Application of the predecessor function proves that $\textsf{suc}$ is injective. This gives us a function of type

$$(\textsf{suc}(m +_{\mathbb{N}} k) = \textsf{suc}(n +_{\mathbb{N}} k)) \to ((m +_{\mathbb{N}} k) = (n +_{\mathbb{N}} k)).$$

Again, by function application, we have reduced our goal to

$$((m +_{\mathbb{N}} \mathsf{suc}(k)) = (n +_{\mathbb{N}} \mathsf{suc}(k))) \to (\mathsf{suc}(m +_{\mathbb{N}} k) = \mathsf{suc}(n +_{\mathbb{N}} k)).$$

Assuming $q : ((m +_{\mathbb{N}} \mathsf{suc}(k)) = (n +_{\mathbb{N}} \mathsf{suc}(k)))$**, we can form a sequence of identifications**

$$\mathsf{suc}(m +_{\mathbb{N}} k) = m +_{\mathbb{N}} \mathsf{suc}(k) = n +_{\mathbb{N}} \mathsf{suc}(k) = \mathsf{suc}(n +_{\mathbb{N}} k),$$

where the outer equalities are judgemental.
Remark: These two proofs are formalized and in the course repository. They're called plus-on-paths and plus-is-injective in the module natural-numbers-functions.