HoTT Chapter 2 Exercises

October 13, 2014

```
{-# OPTIONS --without-K #-}

module Ch2 where

open import Base
open import Ch1
```

1

Lemma (2.1.2)**.** *For every type A and every $x, y, z : A$ there is a function $(x = y) \to (y = z) \to (x = z)$ written $p \to q \to p \cdot q$, such that $\mathsf{refl}_x \cdot \mathsf{refl}_x \equiv \mathsf{refl}_x$ for any $x : A$.*

We call $p \cdot q$ the concatenation or composite of p and q.

Exercise 2.1 Show that the three obvious proofs of Lemma 2.1.2 are pairwise equal.

Proof. (this justifies denoting "the" concatenation function as $\cdot$)

First, we need a type to inhabit. The type of any concatenation operator is

$$\prod_{x,y,z:A} \prod_{p:x=y} \prod_{q:y=z} (x = z)$$

Thus far, the only tool we have to inhabit such a type is path induction. So, we first write down a family

$$D_1(x, y, p) : \prod_{z:A} (y = z) \to (x = z)$$

That is, given $x, y : A$ and a path from x to y, we want a function that takes paths from y to z to paths from x to z.

Path induction dictates that we now need a

$$d_1 : \prod_{x:A} D(x, x, \mathsf{refl}_x)$$

hence

$$d_1(x) : \prod_{z:A} (x = z) \to (x = z)$$

So, given a path from x to z, we want a path from x to z. We'll take the easy way out on this one!

```
_•₁_ : ∀ {i} {A : Type i} {x y z : A} → (x == y) → (y == z) → (x == z)
_•₁_ {i} {A} {_} {_} {z} = ind== D d where
    D : (x y : A) → (p : x == y) → Type i
    D x y _ = y == z → x == z

    d : (x : A) → D x x refl
    d _ = λ q → q
```

For another construction, we do path induction in "the other direction". That is, we will define

$$D_2 : \prod_{y,z:A} \prod_{q:y=z} (x = y) \to (x = z)$$

In other words, given y and z and a path from y to z, we want a function that takes paths from x to y to paths from x to z.

Just like the previous proof, we need a

$$d_2(y) : (y = z) \to (y = z)$$

This is a bit trickier in Agda, because we really want to define a curried function

$$(\bullet_2\ q)\ p = p \bullet q$$

However, we also want the type to be exactly the same as the types of the other constructions. Hence, we will use a twist map.

```
_•₂_ : ∀ {i} {A : Type i} {x y z : A} → (x == y) → (y == z) → (x == z)
_•₂_ = twist concat2 where
    concat2 : ∀ {i} {A : Type i} {x y z : A} → (y == z) → (x == y) → (x == z)
    concat2 {i} {A} {x} {_} {_} = ind== D d where
      D : (y z : A) → (q : y == z) → Type i
      D y z _ = x == y → x == z

      d : (y : A) → D y y refl
      d _ = λ q → q
    twist : ∀ {i} {A : Type i} {x y z : A} → ((y == z) → (x == y) →
        (x == z)) → ((x == y) → (y == z) → (x == z))
    twist f = λ p → λ q → f q p
```

Note that these two constructions use path induction to reduce one side or the other to the "identity" path (in the first case refl_x and in the second case refl_y). We can also do double induction to reduce both p and q to the refl_x and refl_y.

We begin with the same type family as the first proof:

$$D_1 : \prod_{x,y:A} \prod_{p:x=y} (y = z) \to (x = z)$$

but we now wish to find a different inhabitant

$$d_1'(x) : (x = z) \to (x = z)$$

We will use path induction to construct d_1'. We introduce a family:

$$E : \prod_{x,z:A} \prod_{q:x=z} (x = z)$$

we now need

$$e(x) : (x = x)$$

which is gotten quite easily:

$$e(x) = \mathsf{refl}_x$$

```
_•₃_ : ∀ {i} {A : Type i} {x y z : A} → x == y → y == z → x == z
_•₃_ {i} {A} {_} {_} {z} = ind== D d where
    D : (x y : A) → (p : x == y) → Type i
    D x y _ = y == z → x == z

    d : (x₁ : A) → D x₁ x₁ refl
    d _ = ind== E e where
      E : (x z : A) (q : x == z) → Type i
      E x z _ = x == z

      e : (x : A) → E x x refl
      e _ = refl
```

We now want to show that these constructions are pairwise equal. By this, we mean "propositional equality" - hence we must find paths between each pair of constructions.

In each case, we perform a double induction on paths, first reducing p to refl, and then reducing q to refl.

```
•₁=•₂ : ∀ {i} {A : Type i} {x y z : A}
    (p : x == y) (q : y == z) → p •₁ q == p •₂ q
•₁=•₂ {i} {A} {_} {_} {z} = ind== D d where
    D : (x y : A) → x == y → Type i
    D _ y p = (q : y == z) →  p •₁ q == p •₂ q

    d : (x : A) → D x x refl
    d _ = ind== E e where
      E : (y₁ z₁ : A) → y₁ == z₁ → Type i
      E _ _ q = refl •₁ q == refl •₂ q

      e : (x₁ : A) → E x₁ x₁ refl
      e _ = refl

•₂=•₃ : ∀ {i} {A : Type i} {x y z : A}
    (p : x == y) (q : y == z) →  p •₂ q == p •₃ q
•₂=•₃ {i} {A} {x} {y} {z} = ind== D d where
    D : (x y : A) → x == y → Type i
    D _ y p = (q : y == z) → p •₂ q == p •₃ q

    d : (x : A) → D x x refl
    d x = ind== E e where
      E : (y₁ z₁ : A) → y₁ == z₁ → Type i
      E _ _ q = refl •₂ q == refl •₃ q

      e : (x₁ : A) → E x₁ x₁ refl
      e _ = refl -- : concat2' refl refl == concat3' refl refl


•₁=•₃ : ∀ {i} {A : Type i} {x y z : A} (p : x == y) (q : y == z) → p •₁ q == p •₃ q
•₁=•₃ {i} {A} {_} {_} {z} = ind== D d where
    D : (x y : A) → x == y → Type i
    D x y p = (q : y == z) →  p •₁ q == p •₃ q

    d : (x : A) → D x x refl
    d _ = ind== E e where
      E : (y z : A) → (q : y == z) → Type i
      E _ _ q =  refl •₁ q == refl •₃ q

      e : (y : A) → E y y refl
      e _ = refl -- : concat1' refl refl == concat3' refl refl
```

$\square$

2

Lemma (2.2.1)**.** *The three equalities of proofs constructed in the previous exercise form a commutative triangle. In other words, if the three definitions of concatenation are denoted by* $(p \cdot_1 q)$*,* $(p \cdot_2 q)$*, and* $(p \cdot_3 q)$*, then the concatenated equality*

$$(p \cdot_1 q) = (p \cdot_2 q) = (p \cdot_3 q)$$

is equal to the equality

$$(p \cdot_1 q) = (p \cdot_3 q)$$

Proof. Despite the fact that we're working with the somewhat myserious type of "equalities of equalities", this remains a statement about the propositional equality of two paths. The only tool we have for establishing such an equality is path induction.

First, we fix the definition of concatenation:

```
_•_ = _•₁_
```

We must now show that, for all paths p, q, the proof that $p \cdot_1 q$ is equal to $p \cdot_2 q$ followed by the proof that $p \cdot_2 q$ is equal to $p \cdot_3 q$ is equal to the proof that $p \cdot_1 q$ is equal to $p \cdot_3 q$.

This is exactly expressed in the following type signature:

Since the theorem is quantified over two paths, we shall do double path induction. So, it really just boils down to the theorem being true when both p and q are the identity.

```
concat-commutative-triangle : ∀ {i} {A : Type i} {x y z : A} (p : x == y) (q : y == z) →
    (•₁=•₂ p q) • (•₂=•₃ p q) == •₁=•₃ p q

concat-commutative-triangle {i} {A} {_} {_} {z} = ind== D d where
    D : (x y : A) → x == y → Type i
    D _ y p = (q : y == z) →
      (•₁=•₂ p q) • (•₂=•₃ p q) == •₁=•₃ p q

    d : (x : A) → D x x refl
    d _ = ind== E e where
      E : (y z : A) → (q : y == z) → Type i
      E _ _ q =
        (•₁=•₂ refl q) • (•₂=•₃ refl q) == •₁=•₃ refl q

      e : (y : A) → E y y refl
      e _ = refl
```

□

At this point, it might be helpful to review the definitions of the different concatenation functions. In particular, $\mathsf{refl} \cdot \mathsf{refl} \equiv \mathsf{refl}$ where $\cdot$ is any of $\cdot_1$, $\cdot_2$, or $\cdot_3$.

3

4

Define, by induction on n, a general notion of n-dimensional path in a type A, simultaneously with the type of boundaries for such paths.

We'll define n-paths recursively in terms of $n-1$ paths by recursion on $\mathbb{N}$. There are two cases. Given a type A:

A 0-path is an inhabitant of A.

A n-path, for $n > 0$, is an inhabitant of $p = q$ where p and q are $(n-1)$-paths.

I'm going to take two steps and then settle it once and for all!

```
data _==2_ {i} {A : Type i} {a : A} {b : A} (p : a == b) : (a == b) -> Type i where
  refl2 : p ==2 p

data _==3_ {i} {A : Type i} {a : A} {b : A} {p : a == b} {q : a == b}
    (α : p == q) : (p == q) -> Type i where
  refl3 : α ==3 α

npaths : ∀ {i} (A : Type i) -> ℕ -> Type i
npaths {i} A 0 = A
npaths {i} A (S n) = Σ (npaths A n) λ q -> Σ (npaths A n) (λ p → p == q)
```

This is not required by the exercise, but let's define the n-dimensional identity by induction on $\mathbb{N}$. In topology, this would be the constant map from the n-cube to a point.

```
refln : ∀ {i} (A : Type i) (a : A) -> (n : ℕ) -> npaths A n
-- TODO: Fix module stuff so I can write "indN" instead of "Ex1-4.indN"
refln A a = Ex1-4.indN a E where
  E = λ n → λ q → q , (q , refl)
```

Okay, now we have to define a boundary map on npaths. The boundary of an n-path should be a pair of $(n-1)$-paths:

```
boundary : ∀ {i} {A : Type i} {n : ℕ} ->
  (npaths A (S n)) -> (npaths A n) × (npaths A n)
boundary {i} {A} {n} (p , (q , α)) = (p , q)
```