HoTT Chapter 2 Exercises

October 15, 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
```

□

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* $(p \bullet_1 q)$*,* $(p \bullet_2 q)$*, and* $(p \bullet_3 q)$*, then the concatenated equality*

$$(p \bullet_1 q) = (p \bullet_2 q) = (p \bullet_3 q)$$

is equal to the equality

$$(p \bullet_1 q) = (p \bullet_3 q)$$

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 $p \bullet_1 q$ is equal to $p \bullet_2 q$ followed by the proof that $p \bullet_2 q$ is equal to $p \bullet_3 q$ is equal to the proof that $p \bullet_1 q$ is equal to $p \bullet_3 q$.

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
```

$\square$

At this point, it might be helpful to review the definitions of the different concatenation functions. In particular, $\mathsf{refl} \centerdot \mathsf{refl} \equiv \mathsf{refl}$ where $\centerdot$ is any of $\centerdot_1$, $\centerdot_2$, or $\centerdot_3$.

3

4

Let's try to do it one stage at a time:

- a 0-path is a point in A.
- a 1-path is a path between 0-paths.

```
0-path : ∀ {i} (A : Type i) -> Type i
0-path A = A

1-path : ∀ {i} (A : Type i) -> Type i
1-path A = Σ (0-path A) (λ a -> Σ (0-path A) (λ b -> (a == b)))

-- the boundary of a 1-path is a pair of 0-paths
δ₁ : ∀ {i} {A : Type i} -> (p : 1-path A) -> (0-path A) × (0-path A)
δ₁ (a , (b , p)) = (a , b)
```

The boundary of a 0-path is somewhat mysterious, so we shall leave it undefined.

We now would like to define a 2-path as a path between 1-paths. However, two arbitrary 1-paths look like this:

$$a \xrightarrow{p} b$$

$$a' \xrightarrow{q} b'$$

That is, p and q are paths between different points. Hence, a path between p and q doesn't make sense. That is, it's not well typed.

However, suppose we have paths x, y as follows:

$$\begin{tikzcd} a \arrow[r, "p"] \arrow[d, "x"] & b \arrow[d, "y"] \\ a' \arrow[r, "q"] & b' \end{tikzcd}$$

It would certainly make sense to ask for a path of type $p \cdot y = x \cdot q$.

So, it seems that to define 2-paths, we need pairs of 1-paths together with vertical paths like x and y above. So we'll define it as a Σ-type:

$$\Sigma_{p,q:\text{1-paths}}\Sigma_{x:\text{src p = src q}}\Sigma_{\text{dst p = dst q}}p \cdot y = x \cdot q$$

We will write down some helper functions and then formalize this:

```
-- Some convenience functions!
src : ∀ {i} {A : Type i} {a : A} {b : A} -> a == b -> A
src {_} {_} {a} {_} p = a

dst : ∀ {i} {A : Type i} {a : A} {b : A} -> a == b -> A
dst {_} {_} {_} {b} p = b

map : ∀ {i} {A : Type i} (p : (1-path A)) -> (fst p) == (fst (snd p))
map p = snd (snd p)

2-path : ∀ {i} (A : Type i) -> Type i
2-path A = Σ (1-path A) λ p -> Σ (1-path A) λ q →
  Σ ((src (map p)) == (src (map q))) λ x → Σ ((dst (map p)) == (dst (map q))) λ y →
    x · (map q) == (map p) · y

-- The boundary of a 2 path as a pair of 1 paths
δ2 : ∀ {i} {A : Type i} -> 2-path A -> (1-path A) × (1-path A)
δ2 {i} {A} (p , (q , (x , (y , α)))) = p , q
```

A boundary of a 2-path can be thought of as a loop. We can formalize this:

```
-- Definition of inverses. This should be put somewhere else.
inverse : ∀ {i} {A : Type i} {a : A} {b : A} -> a == b -> b == a
inverse {i} {A} = ind== D d where
  D : (a b : A) (p : a == b) → Type i
  D a b _ = b == a
  d : (x : A) → D x x refl
  d _ = refl

-- Just to be cute - the boundary of a 2 path as a loop
δ2-loop : ∀ {i} {A : Type i} -> 2-path A -> 1-path A
δ2-loop (p , (q , (x , (y , α)))) =
  (src (map p) , (src (map p) ,
    ((x · (map q)) · (inverse y)) · (inverse (map p))))
```

If one tries to continue in this manner, the Σ-types will become rather large! So it would be nice to appeal to some kind of recursion at this point.

Luckily, it turns out that equality of inhabitants of Σ-types contain all the lower dimensional equalities to make this work!

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

δ : ∀ {i} {A : Type i} -> (n : ℕ) ->
  (n-path {i} {A} (S n)) -> (n-path {i} {A} n) × (n-path {i} {A} n)
δ n p = fst p , fst (snd p)
```

This is not evidently geometric. To make the connection, we need to use some facts about equalities of sigma types.