

# HoTT Chapter 2 Exercises

October 20, 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  $(x = y) \rightarrow (y = z) \rightarrow (x = z)$  written  $p \rightarrow q \rightarrow p \bullet q$ , such that  $\text{refl}_x \bullet \text{refl}_x \equiv \text{refl}_x$  for any  $x : A$ .*

*We call  $p \bullet q$  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  $\bullet$ )

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

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

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

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

hence

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

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} {x} {y} {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

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

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

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

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

$$(\cdot_2 q) p = p \cdot q$$

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} {y} {z} = 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  $\text{refl}_x$  and in the second case  $\text{refl}_y$ ). We can also do double induction to reduce both  $p$  and  $q$  to the  $\text{refl}_x$  and  $\text{refl}_y$ .

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

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

but we now wish to find a different inhabitant

$$d'_1(x) : (x = z) \rightarrow (x = z)$$

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:

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

```

_•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  $\text{refl}$ , and then reducing  $q$  to  $\text{refl}$ .

```

•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} {x} {y} {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} {x} {y} {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 mysterious 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
```

□

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

### 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:

$$\begin{array}{ccc} a & \xrightarrow{p} & b \\ \downarrow x & & \downarrow y \\ a' & \xrightarrow{q} & b' \end{array}$$

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  $\Sigma$ -type:

$$\sum_{p,q} \sum_{x:src(p)=src(q)} \sum_{y:dst(p)=dst(q)} p \cdot y = x \cdot q$$

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 {i} {A} {a} {b} p = a

dst : ∀ {i} {A : Type i} {a : A} {b : A} -> a == b -> A
dst {i} {A} {a} {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  $\Sigma$ -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  $\Sigma$ -types contain all the lower dimensional equalities to make this work!

```

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

δ : ∀ {i} {A : Type i} -> (n : ℕ) ->
  (n-path {i} {A} (S n)) -> (n-path n) × (n-path 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.

```

Σ-path-trans : ∀ {i} {A : Type i} {P : A -> Type i}
  -> {w : Σ A P} -> {w' : Σ A P} -> w == w'
  -> (Σ ((fst w) == (fst w')) λ p
    -> (transport {i} {i} {A} P p (snd w)) == (snd w'))
Σ-path-trans {i} {A} {P} = ind== D d where
  D : (w : Σ A P) -> (w' : Σ A P) -> w == w' -> Type i
  D w w' _ = (Σ ((fst w) == (fst w')) λ p
    -> (transport {i} {i} {A} P p (snd w)) == (snd w'))
  d : (w : Σ A P) -> D w w refl
  d _ = refl , refl

```

**Lemma.** *If  $n > 1$ , then the boundary of an  $n$ -path is a closed  $(n - 1)$ -path.*

*Proof.* The following code corresponds to this diagram:

$$\begin{array}{ccc}
 a & \xrightarrow{p} & b \\
 \downarrow x & & \\
 a' & \xrightarrow[p]{p'} & b'
 \end{array}$$

```

δ' : ∀ {i} {A : Type i} -> {n : ℕ}
  -> (n-path {i} {A} (S (S n))) -> (n-path {i} {A} (S n))
δ' {i} {_} {n} α = a' , ( a' , q • (inverse p') ) where
  -- first, unpack α
  a : n-path n
  a = fst (fst α)
  b : n-path n
  b = fst (snd (fst α))
  p : a == b
  p = snd (snd (fst α))

  a' : n-path n

```



```

a' = fst (fst (snd α))
b' : n-path n
b' = fst (snd (fst (snd α)))
q : a' == b'
q = snd (snd (fst (snd α)))

-- Apply first sigma equality
P : n-path n -> Type i
P p = Σ (n-path n) λ q -> p == q
t : Σ (a == a') λ x -> (transport P x (b , p)) == (b' , q)
t = Σ-path-trans (snd (snd α))
x : a == a'
x = fst t

-- Apply second (nested) sigma equality
P' : n-path n -> Type i
P' c = a' == c
y' : Σ (n-path n) λ c -> a' == c
y' = transport P x (b , p)
t' : y' == (b' , q)
t' = snd t
t'' : Σ ((fst y') == b') λ y -> (transport P' y (snd y')) == q
t'' = Σ-path-trans t'
y : fst y' == b'
y = fst t''
α' : (transport P' y (snd y')) == q
α' = snd t''
p' : a' == b'
p' = src α'

```

□

## 5

## 6

**Lemma.** *Let  $p : x = y$  in  $A$ . Then for all  $z : A$ , the function  $p \bullet - : (y = z) \rightarrow (x = z)$  is an equivalence.*

*Proof.* An obvious candidate for a quasi-inverse would be a map that concatenates with  $\text{inverse}(p)$ .

We need to use the groupoid laws for  $\bullet$ , as well as whiskering over higher paths.

Here are some groupoid laws:

```

--inv-l : ∀ {i} {A : Type i} {a b : A} -> (p : a == b) -> ((inverse p) · p) == refl
--inv-l {i} {A} {a} {b} = ind== D d where
  D : (a b : A) -> (p : a == b) → Type i
  D _ _ p = ((inverse p) · p) == refl
  d : (x : A) → D x x refl
  d _ = refl

--inv-r : ∀ {i} {A : Type i} {a b : A} -> (p : a == b) -> p · (inverse p) == refl
--inv-r {i} {A} {a} {b} = ind== D d where
  D : (a b : A) -> (p : a == b) → Type i
  D _ _ p = p · (inverse p) == refl
  d : (x : A) → D x x refl
  d _ = refl

--func : ∀ {i} {A : Type i} {a : A} {b : A} {c : A} -> (p : a == b) -> (b == c) -> (a == c)
--func p q = p · q

--id-l : ∀ {i} {A : Type i} {a b : A} -> (p : a == b) -> refl · p == p
--id-l {i} {A} {a} {b} = ind== D d where
  D : (a b : A) -> (p : a == b) → Type i
  D _ _ p = (refl · p) == p
  d : (x : A) → D x x refl
  d _ = refl

--id-r : ∀ {i} {A : Type i} {a b : A} -> (p : a == b) -> p · refl == p
--id-r {i} {A} {a} {b} = ind== D d where
  D : (a b : A) -> (p : a == b) → Type i
  D _ _ p = p · refl == p
  d : (x : A) → D x x refl
  d _ = refl

--assoc : ∀ {i} {A : Type i} {w x y z : A}
  -> (p : w == x) -> (q : x == y) -> (r : y == z)
  -> p · (q · r) == (p · q) · r
--assoc {i} {A} {w} {x} {y} {z} = ind== D d where
  D : (w x : A) -> (p : w == x) -> Type i
  D _ x p = (q : x == y) -> (r : y == z) -> p · (q · r) == (p · q) · r
  d : (x : A) -> D x x refl
  d _ = ind== E e where
    E : (x y : A) -> (q : x == y) -> Type i
    E _ y q = (r : y == z) -> refl · (q · r) == (refl · q) · r
    e : (x : A) -> E x x refl
    e _ = ind== F f where
      F : (y z : A) -> (r : y == z) -> Type i
      F _ z r = refl · (refl · r) == (refl · refl) · r

```

```

f : (x : A) -> F x x refl
f _ = refl

```

And whiskering for 2-paths (really,  $n + 2$  paths...)

```

whisk-r : ∀ {i} {A : Type i} {x y z : A} {p p' : x == y} {q : y == z}
  -> (p == p') -> ((p · q) == (p' · q))
whisk-r {i} {_} {x} {y} {z} {_} {_} {q} = ind== D d where
  D : (p p' : x == y) -> (α : p == p') -> Type i
  D p p' α = ((p · q) == (p' · q))
  d : (p : x == y) -> D p p refl
  d _ = refl

whisk-l : ∀ {i} {A : Type i} {x y z : A} {p : x == y} {q q' : y == z}
  -> (q == q') -> ((p · q) == (p · q'))
whisk-l {i} {_} {x} {y} {z} {p} {_} {_} = ind== D d where
  D : (q q' : y == z) -> (β : q == q') -> Type i
  D q q' β = ((p · q) == (p · q'))
  d : (q : y == z) -> D q q refl
  d _ = refl

```

We now define the quasi inverse to  $p \cdot -$  as  $p^{-1} \cdot -$ . To do this, we need homotopies from  $(p \cdot -) \circ (p^{-1} \cdot -) \sim id$  and  $(p^{-1} \cdot -) \circ (p \cdot -) \sim id$ . By definition,  $(p \cdot -) \circ (p^{-1} \cdot -) \equiv (p \cdot p^{-1} \cdot -)$ , so we really just need a 2-path  $p \cdot p^{-1} = refl$  (and a 2-path for the symmetric case). This follows from the groupoid laws above:

```

--qinv : ∀ {i} {A : Type i} {x y z : A}
  -> (p : x == y) -> (q-inv (_·_ {i} {A} {x} {y} {z} p))
--qinv p = _·_ (inverse p) ,
  ( (λ q -> ((-·-assoc p (inverse p) q) · whisk-r (-·-inv-r p)) · (-·-id-l q))
    , (λ q -> ((-·-assoc (inverse p) p q) · whisk-r (-·-inv-l p)) · (-·-id-l q)))

```

□