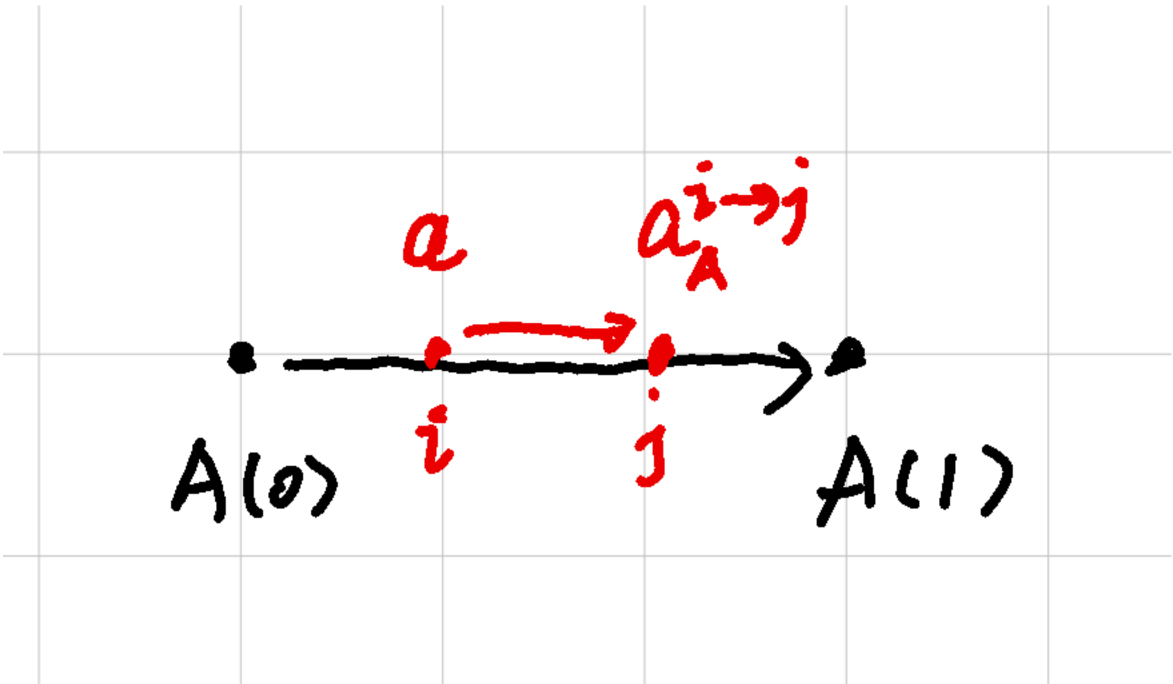
Cubical Agda Explore

Week 3, Spring 2023

Chenchao Ding, Jan 29

Transport

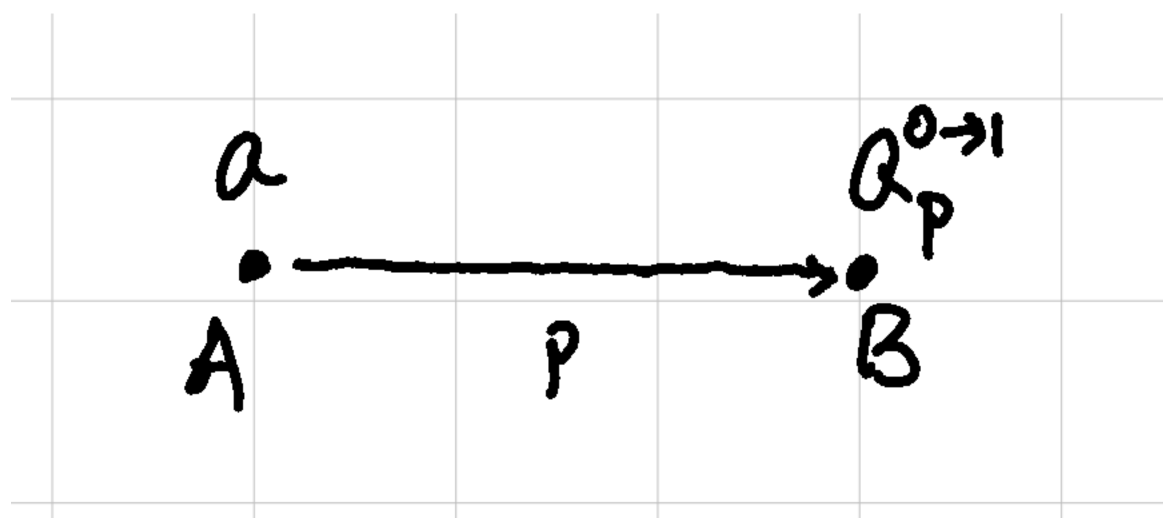
`A : I -> U`



continuous deformation/transformation

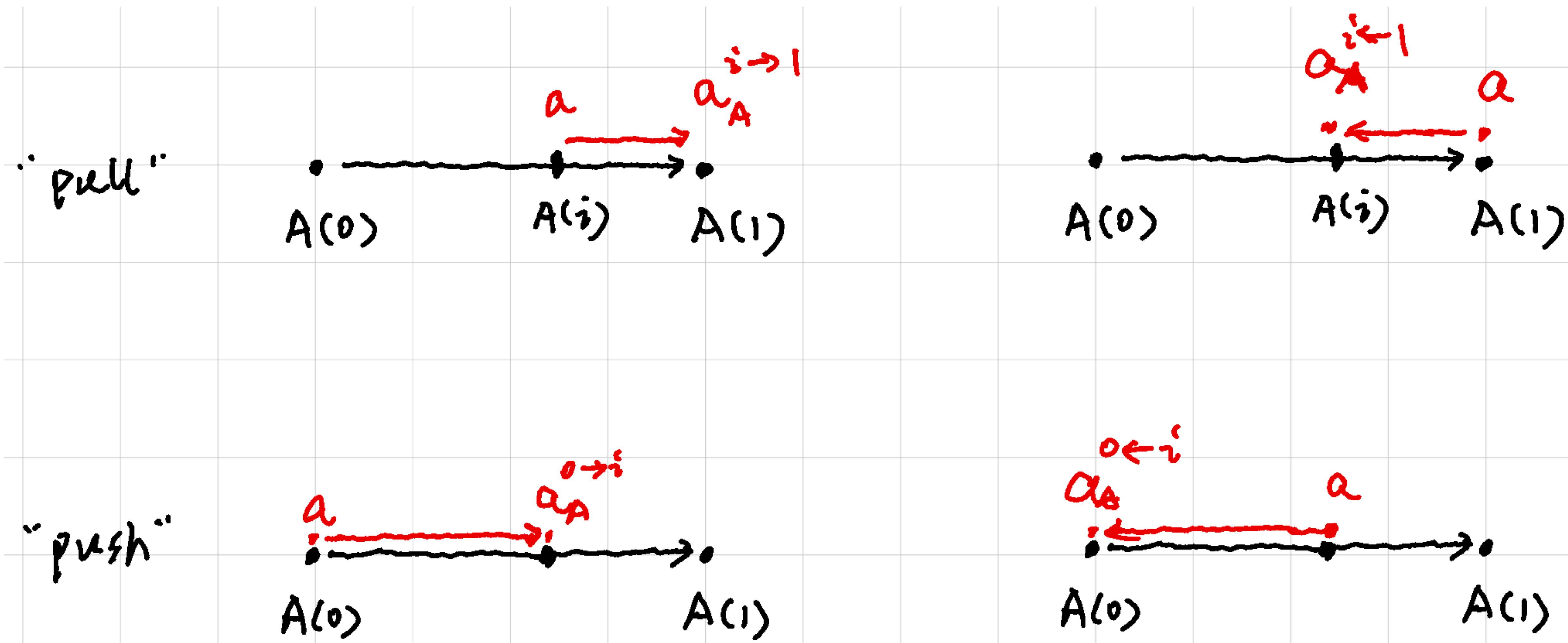
Transport

```
transport : A = B -> A -> B
```



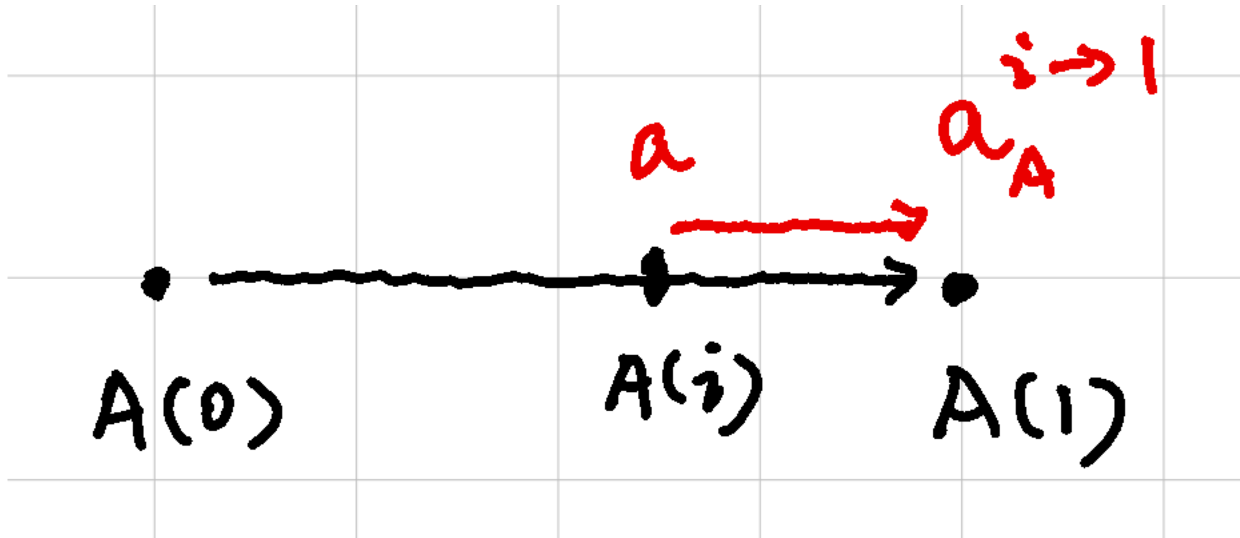
path is like operational semantics ;
transport is like “evaluation”

Transport



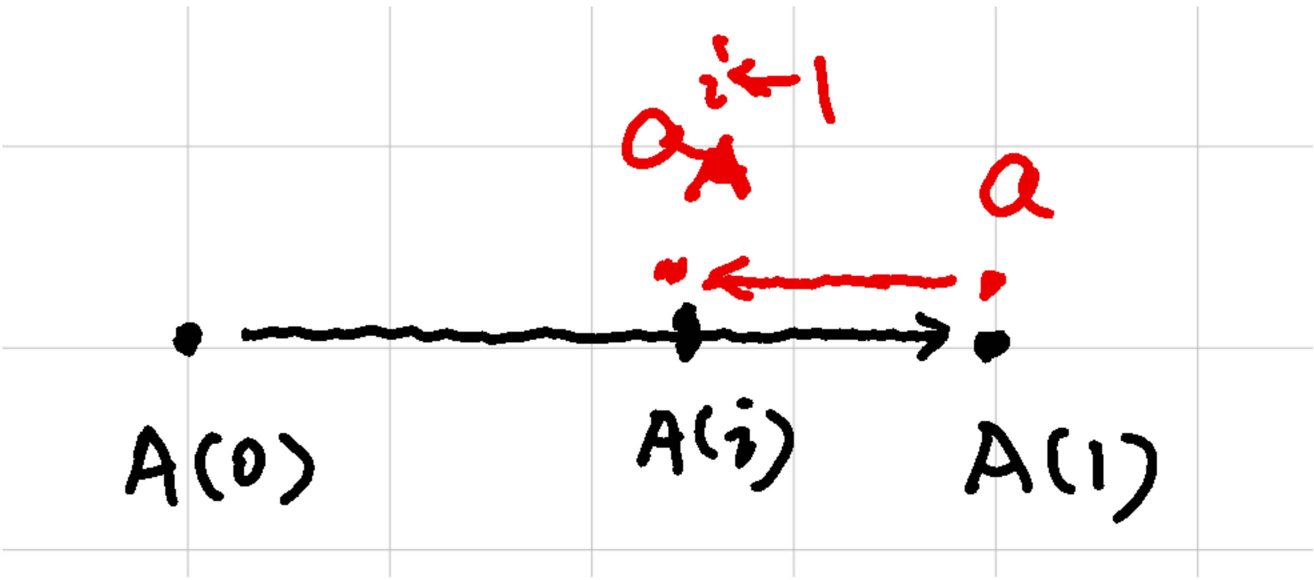
`Cubical.Foundations.CartesianKanOps`

Transport



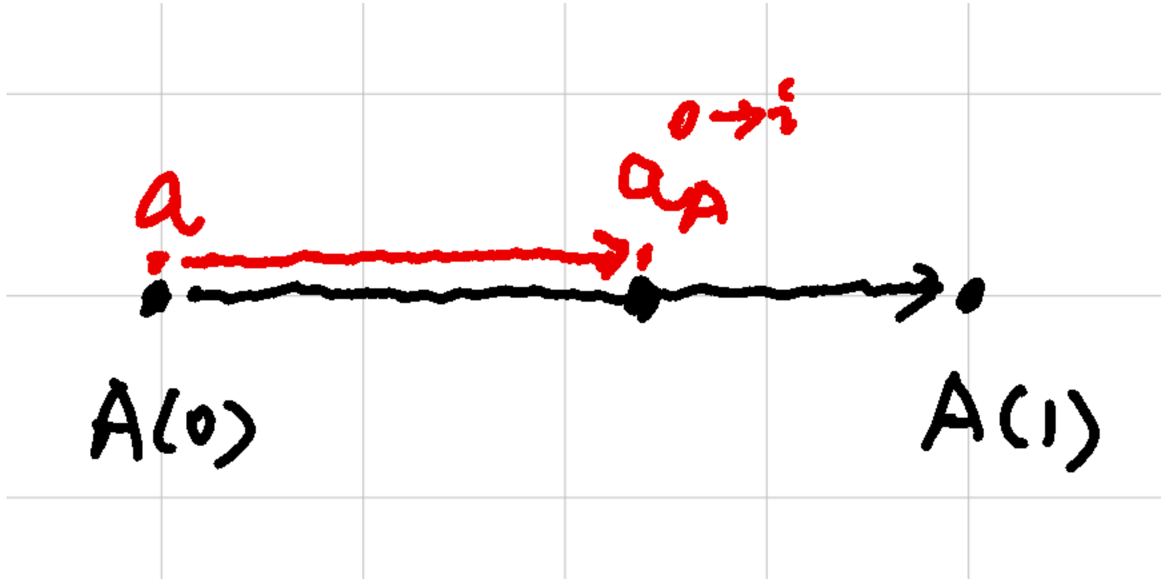

```
pull : ∀ {ℓ} (A : I → Type ℓ) (i : I) → A i → A i1
pull A i a = transp (λ j → A (i ∨ j)) i a
-- i ? 0 --> i
-- i ? 1 --> 1
```

Transport



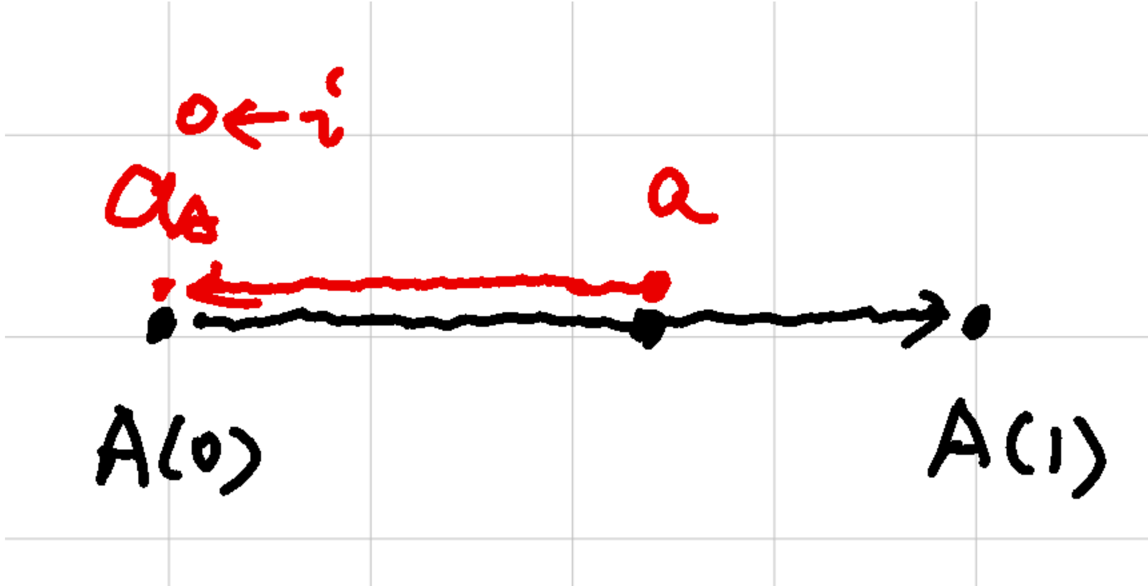

```
pull' : ∀ {ℓ} (A : I → Type ℓ) (i : I) → A i1 → A i
pull' A i a = transp (λ j → A (i ∨ ~ j)) i a
-- i ? 1 --> 1
-- i ? 0 --> i
```

Transport



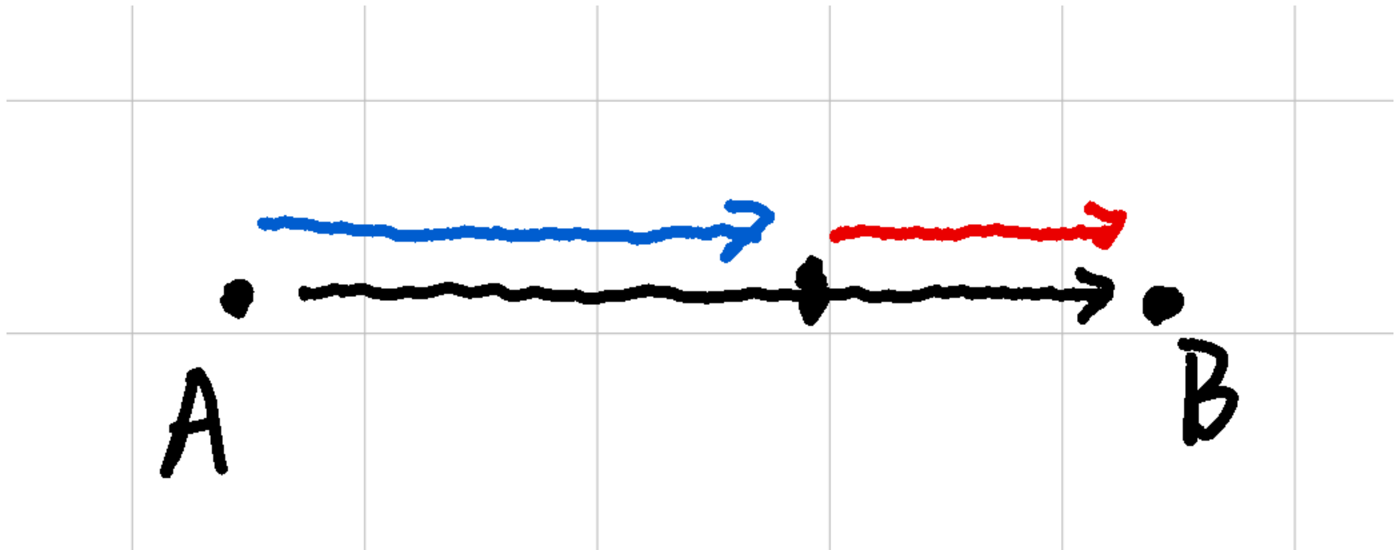

```
push : ∀ {ℓ} (A : I → Type ℓ) (i : I) → A i0 → A i
push A i a = transp (λ j → A (i ∧ j)) (~ i) a
-- i ? 0 --> 0
-- i ? 1 --> i
```

Transport



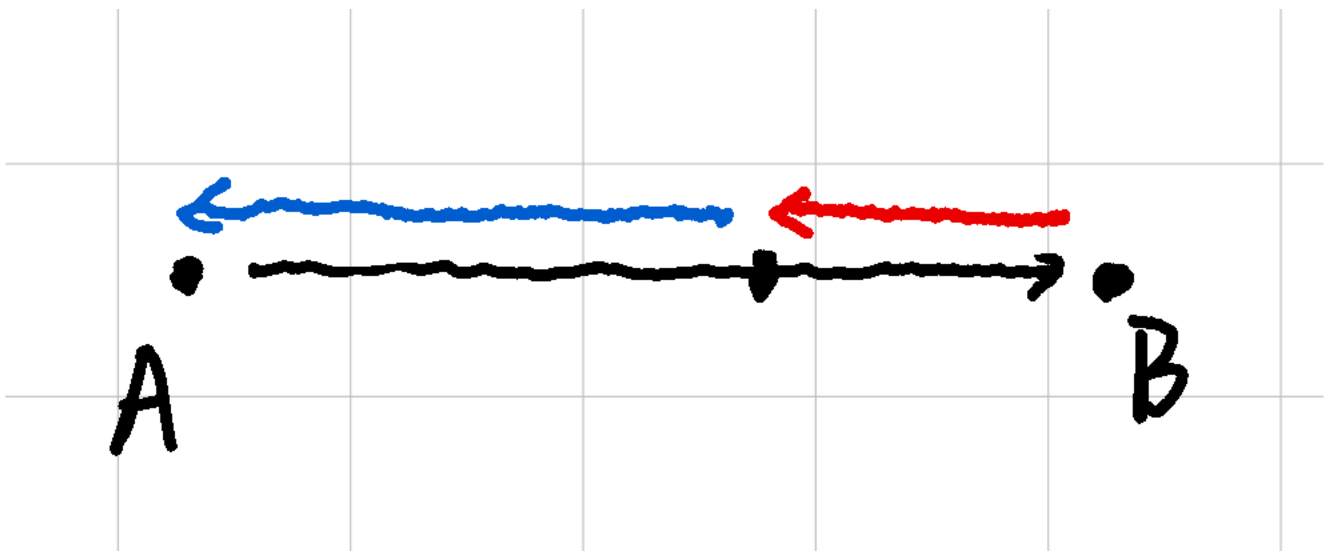

```
push' : ∀ {ℓ} (A : I → Type ℓ) (i : I) → A i → A i0
push' A i a = transp (λ j → A (i ∧ ~ j)) (~ i) a
-- i ? 1 --> i
-- i ? 0 --> 0
```

2 maps from a path




```
sqrt→ : ∀ {ℓ} {A B : Type ℓ} (p : A ≡ B) (i : I)
      → (p i0 → p i) × (p i → p i1)
sqrt→ p i = (λ a → transp (λ j → p (i ∧ j)) i0 a)
          , (λ a → transp (λ j → p (i ∨ j)) i0 a)
```

2 maps from a path




```
sqrt← : ∀ {ℓ} {A B : Type ℓ} (p : A ≡ B) (i : I)
      → (p i1 → p i) × (p i → p i0)
sqrt← p i = (λ b → transp (λ j → p (i ∨ ~ j)) i0 b)
          , (λ b → transp (λ j → p (i ∧ ~ j)) i0 b)
```

2 paths from a path

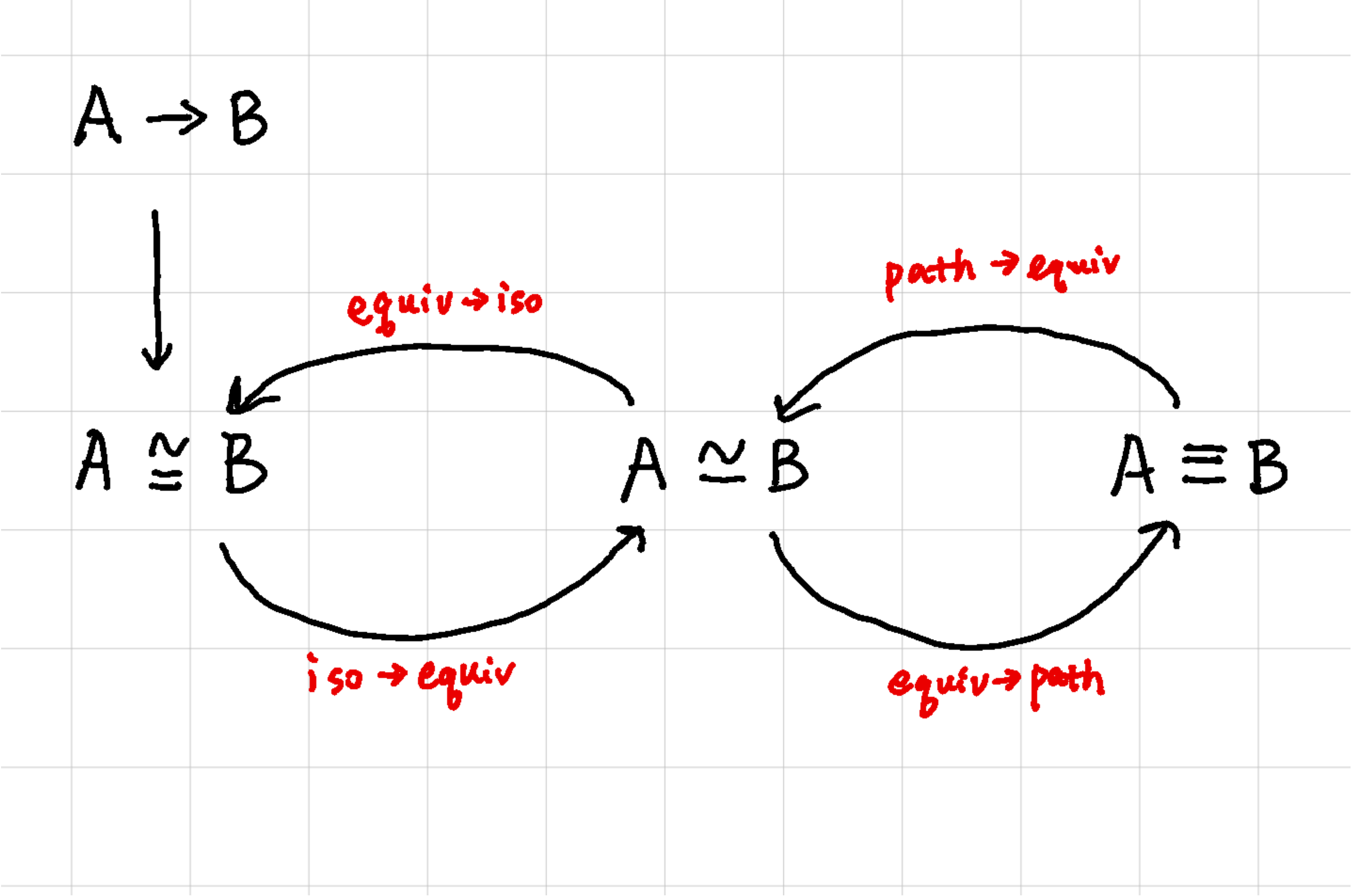
```
pullp : ∀ {ℓ} {A B : Type ℓ} (p : A ≡ B) (i : I)
      → p i ≡ p i1
pullp p i = ua (pulle p i)

pushp : ∀ {ℓ} {A B : Type ℓ} (p : A ≡ B) (i : I)
      → p i0 ≡ p i
pushp p i = ua (pushe p i)

sqrt≡ : ∀ {ℓ} {A B : Type ℓ} (p : A ≡ B) (i : I)
      → (p i0 ≡ p i) × (p i ≡ p i1)
sqrt≡ p i = pushp p i , pullp p i
```

Cubical.Foundations.Univalence
Cubical.Foundations.Isomorphism
Cubical.Foundations.Equiv

2 paths from a path

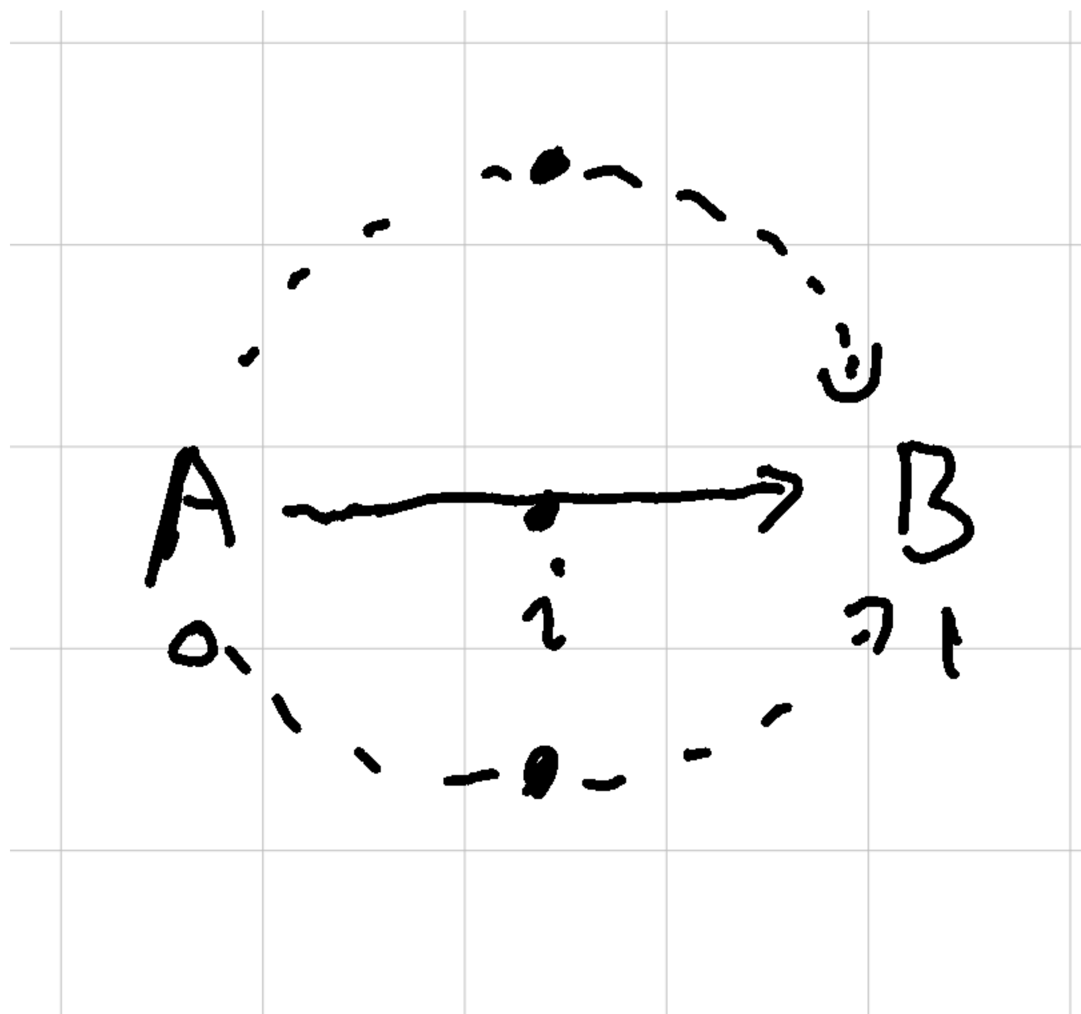


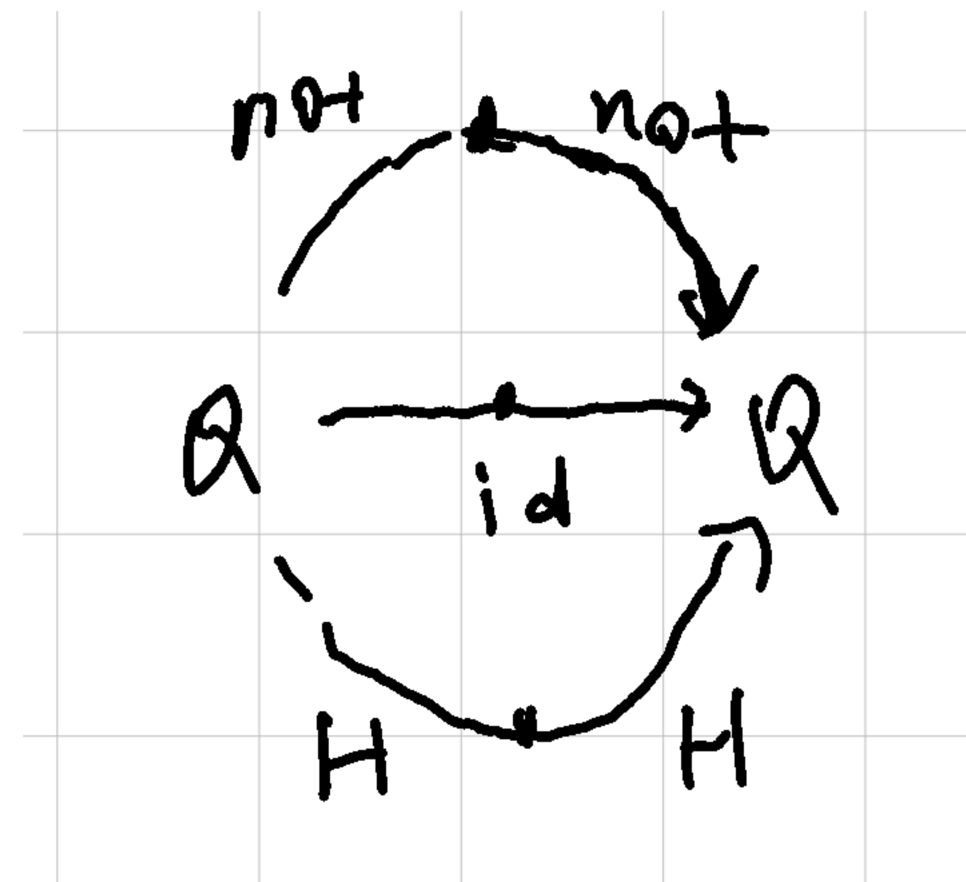
2 paths from a path

```
-- rename transpEquiv from Cubical.Foundations.Transport
pulle : ∀ {ℓ} {A B : Type ℓ} (p : A ≡ B) (i : I) → p i ≃ p i1
pulle P i .fst = transp (λ j → P (i ∨ j)) i
pulle P i .snd
  = transp (λ k → isEquiv (transp (λ j → P (i ∨ (j ∧ k))) (i ∨ ~ k)))
      i (idIsEquiv (P i))
```

```
pushe : ∀ {ℓ} {A B : Type ℓ} (p : A ≡ B) (i : I) → p i0 ≃ p i
pushe P i .fst = transp (λ j → P (i ∧ j)) (~ i)
pushe P i .snd = magic
```

“Intensional” vs. “Extensional”





What do we care about?

“Intensional” vs. “Extensional”

Is this a valid question?

1. The process itself (intensional)
2. Endpoints (extensional)

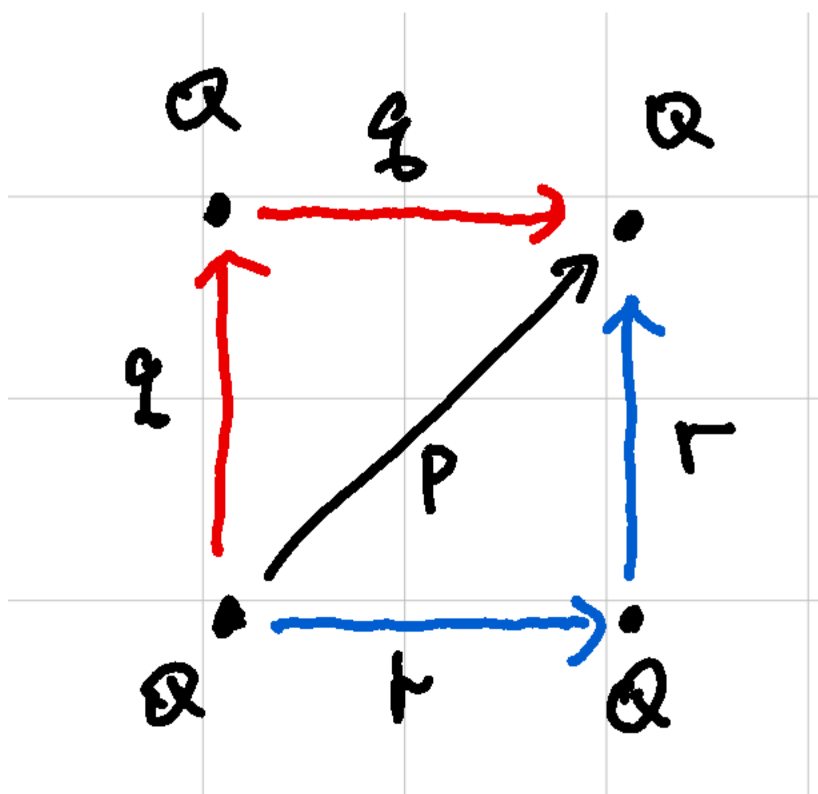
For every path p, there is another path q so that

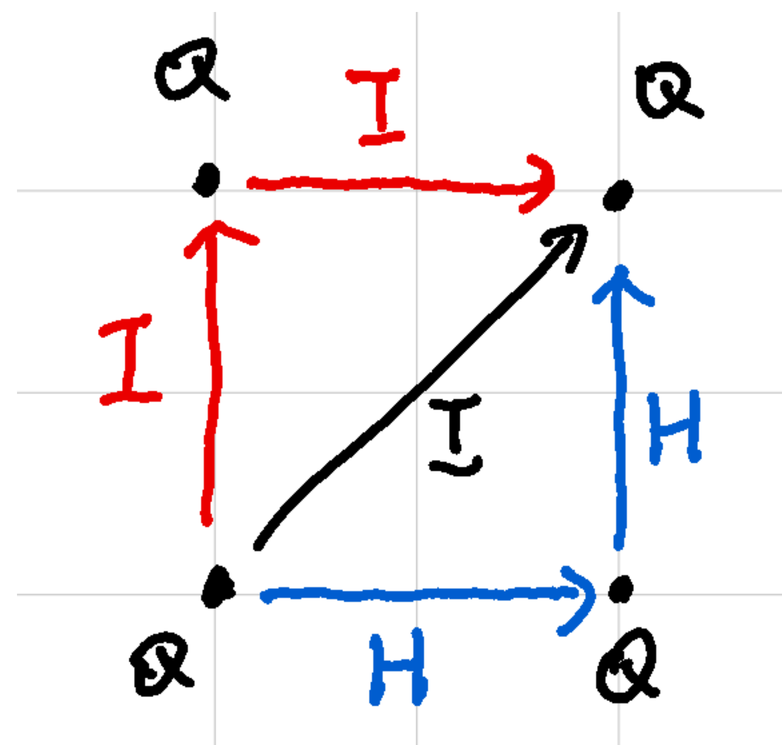
```
transport p = transport q
```

then

```
(sqrt q)
```

Construct from diagonals

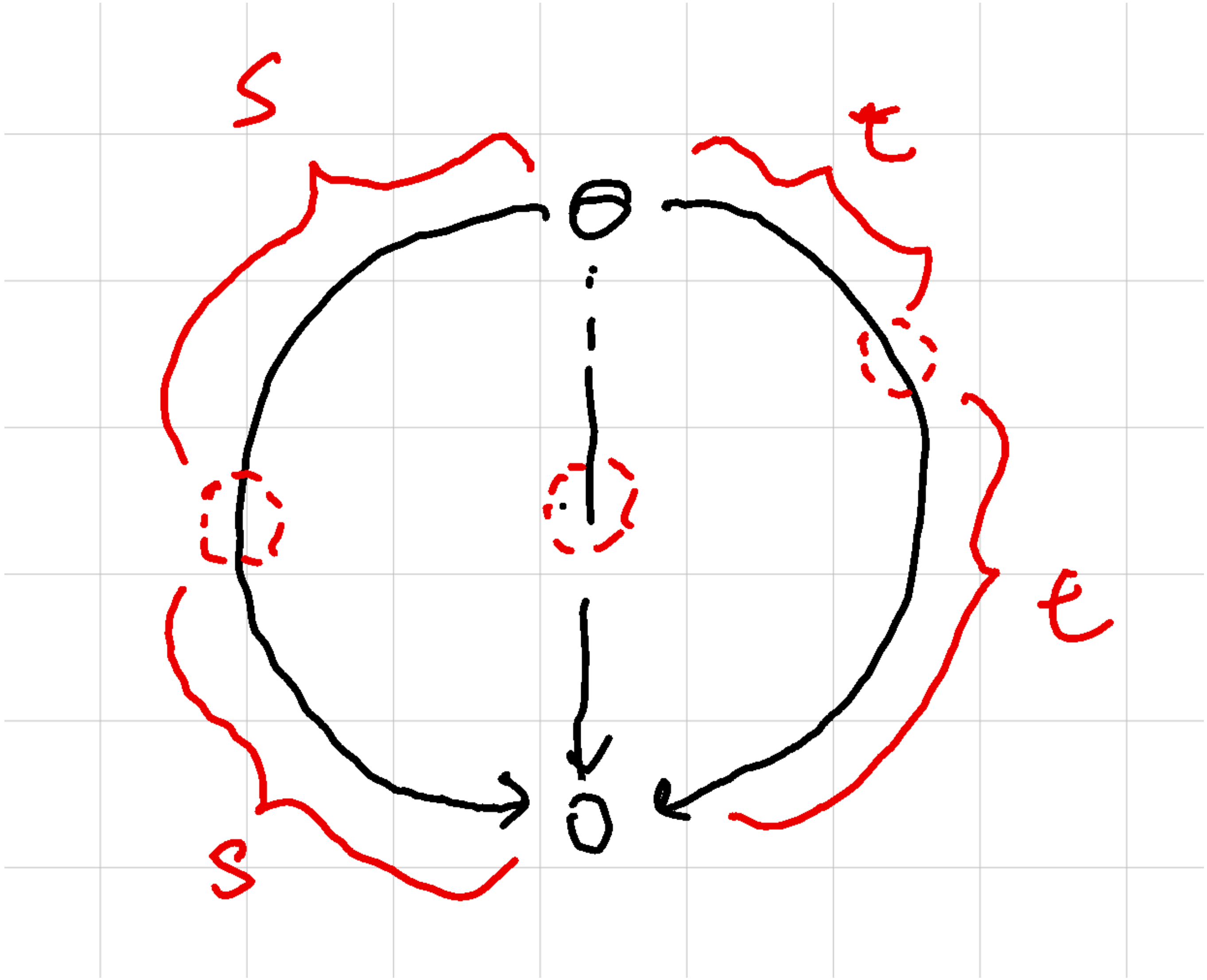




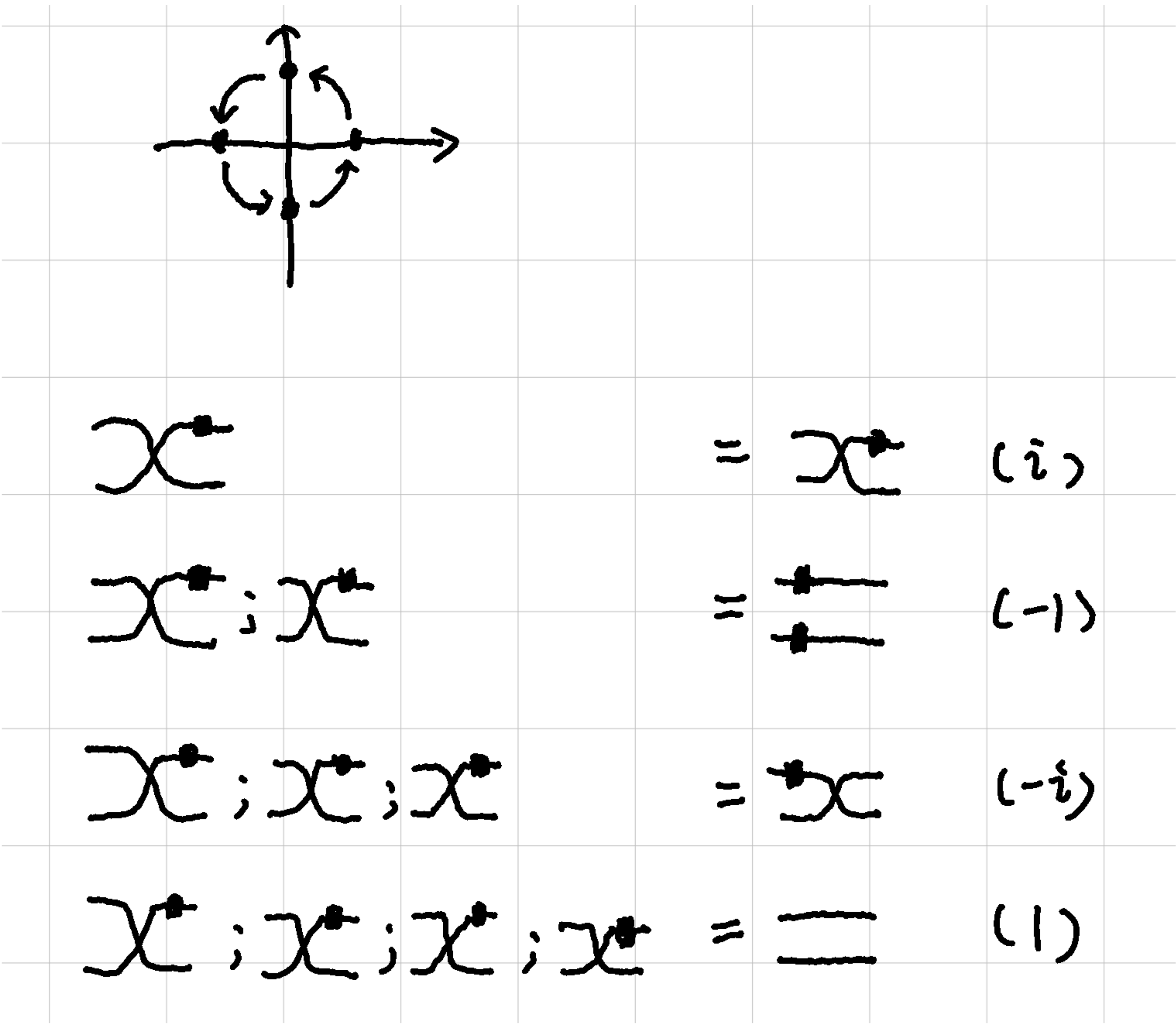
possible… need further exploration

Another question…

`sqrt` with respect to?



Diagrammatic stuff



Can we draw some inspiration?

Square root of matrix?

https://en.wikipedia.org/wiki/Square_root_of_a_2_by_2_matrix