Cubical Agda Explore

Week4, Spring 2023

Chenchao Ding, Feb 5

Baby Language Π_2 : Syntax

```
data Π₂ : Type where
  𝔹 : Π₂
```

Baby Language Π_2 : Syntax

```
data _⟷_ : (A B : Π₂) → Type where

  id₁    : {A : Π₂} → (A ⟷ A)
  not₁   : 𝔹 ⟷ 𝔹
  !₁_    : {A B : Π₂} → (A ⟷ B) → (B ⟷ A)
  _⊙_    : {A B C : Π₂} → (A ⟷ B) → (B ⟷ C) → (A ⟷ C)
  sqrt   : {A : Π₂} → (c : A ⟷ A) → (A ⟷ A)
```

1-combinators

Baby Language Π_2 : Syntax

```
data _⇔_ : {A B : Π₂} (p q : A ↔ B) → Type where

  id₂    : {A B : Π₂} {c : A ↔ B} → c ⇔ c
  !₂_    : {A B : Π₂} {p q : A ↔ B} → (p ⇔ q) → (q ⇔ p)
  _⊙₂_   : {A B : Π₂} {p q r : A ↔ B} → (p ⇔ q) → (q ⇔ r) → (p ⇔ r)

  !id₁   : {A : Π₂} → !₁ id₁{A} ⇔ id₁{A}
  !not₁  : !₁ not₁ ⇔ not₁
```

2-combinators

Baby Language Π_2 : Syntax

```
idl⊙l : {A B : Π₂} {c : A ⇔ B} → (id₁ ⊙ c) ⇔ c
idr⊙l : {A B : Π₂} {c : A ⇔ B} → (c ⊙ id₁) ⇔ c

!r    : {A B : Π₂} (p : A ⇔ B) → p ⊙ !₁ p ⇔ id₁
!l    : {A B : Π₂} (p : A ⇔ B) → !₁ p ⊙ p ⇔ id₁

!!    : {A B : Π₂} {p : A ⇔ B} → !₁ (!₁ p) ⇔ p
`!    : {A B : Π₂} {p q : A ⇔ B} → (p ⇔ q) → (!₁ p ⇔ !₁ q)
```

2-combinators

Baby Language Π_2 : Syntax

```
sqd   : {A : Π₂} {c : A ⇔ A} → sqrt c ⊙ sqrt c ⇔ c
sqf   : {A : Π₂} {c : A ⇔ A} → sqrt (c ⊙ c) ⇔ sqrt c ⊙ sqrt c
sqi   : {A : Π₂} {p q : A ⇔ A} → (p ⇔ q) → sqrt p ⇔ sqrt q
sqc   : {A : Π₂} {c : A ⇔ A} → sqrt c ⊙ c ⇔ c ⊙ sqrt c -- derivable
```

`sqrt` related 2-combinators

Baby Language Π_2 : Semantics

```
⟦_⟧ : Π₂ → Type
⟦ 𝔹 ⟧ = Bool
```

Baby Language Π_2 : Semantics

```
id-path : {T : Type} → T ≡ T
id-path = refl

not-path : Bool ≡ Bool
not-path = isoToPath (iso not not rem rem)
  where
    rem : (b : Bool) → not (not b) ≡ b
    rem false = refl
    rem true  = refl
```

Baby Language Π_2 : Semantics

```
_⟦_⟧₁ : {A B : Π₂} (i : I) (c : A ↔ B) → ⟦ A ⟧ ≡ ⟦ B ⟧
i ⟦ id₁ ⟧₁     = id-path
i ⟦ not₁ ⟧₁    = not-path
i ⟦ !₁ c ⟧₁    = sym (i ⟦ c ⟧₁)
i ⟦ p ⊙ q ⟧₁   = (i ⟦ p ⟧₁) · (i ⟦ q ⟧₁)
i ⟦ sqrt c ⟧₁ = { }1 -- need a semantics model
```

Denotational semantics for 1-combinators (`c2path`)

Baby Language Π_2 : Semantics

```
_⟦_⟧₂ : {A B : Π₂} (i : I) {p q : A ⇔ B}
      → (p ⇔ q) → (i ⟦ p ⟧₁) ≡ (i ⟦ q ⟧₁)
i ⟦ id₂ ⟧₂         = refl
i ⟦ !₂ t ⟧₂        = sym (i ⟦ t ⟧₂)
i ⟦ t₁ ⊙₂ t₂ ⟧₂ = (i ⟦ t₁ ⟧₂) · (i ⟦ t₂ ⟧₂)
i ⟦ sqd ⟧₂         = { }2
i ⟦ sqf ⟧₂         = { }3
i ⟦ sqi t ⟧₂       = { }4
i ⟦ sqc ⟧₂         = { }5
i ⟦ idl⊙l ⟧₂       = sym lUnitT             -- refl · p ≡ p
i ⟦ idr⊙l ⟧₂       = sym rUnitT             -- p · refl ≡ p
i ⟦ !r p ⟧₂        = rCancelT (i ⟦ p ⟧₁)  -- p · (sym p) ≡ refl
i ⟦ !l p ⟧₂        = lCancelT (i ⟦ p ⟧₁)  -- (sym p) · p ≡ refl
i ⟦ assoc⊙l ⟧₂    = assocT
i ⟦ assoc⊙r ⟧₂    = sym assocT
i ⟦ t₁ ⊡ t₂ ⟧₂    = (i ⟦ t₁ ⟧₂) ■ (i ⟦ t₂ ⟧₂) -- p ≡ q → r ≡ s
i ⟦ !id₁ ⟧₂        = refl
i ⟦ !not₁ ⟧₂       = !notp=notp             -- (sym not-path) ≡ no
i ⟦ !! ⟧₂          = refl
i ⟦ `! t ⟧₂        = cong sym (i ⟦ t ⟧₂)
```

Denotational semantics for 2-combinators

GroupoidLawT.agda

```
rUnitT : ∀ {ℓ} {A B : Type ℓ} {p : A ≡ B} → p ≡ p · refl
rUnitT {ℓ}{A}{B}{p} j i = hfill walls (inS (p i)) j
  where
    walls : ∀ j → Partial (~ i ∨ i) (Type ℓ)
    walls j (i = i0) = A
    walls j (i = i1) = B
```

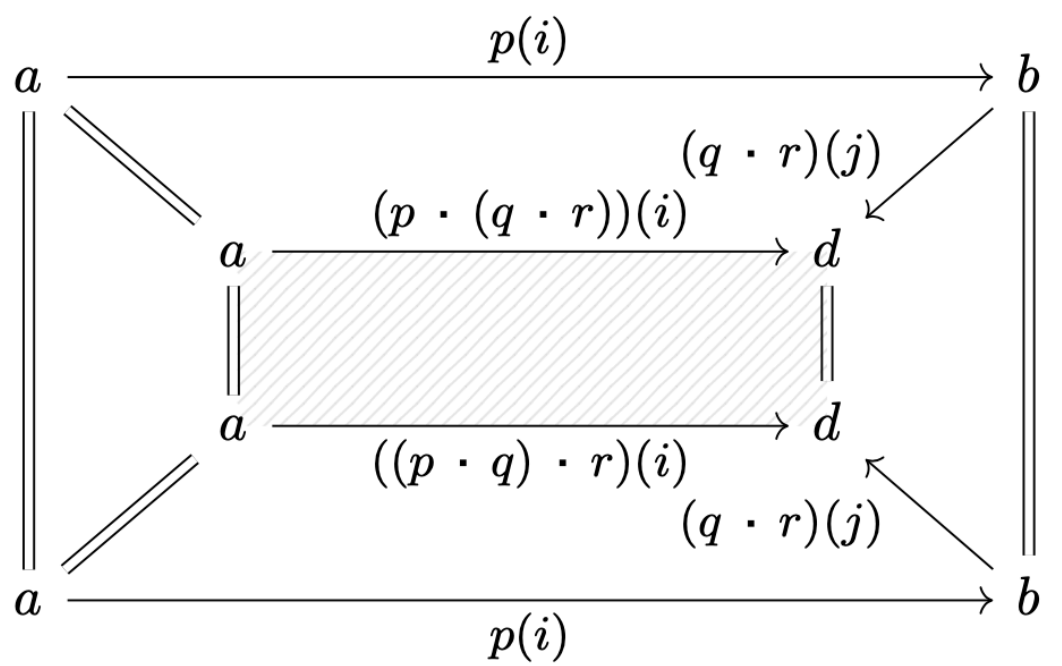
```
lUnitT : ∀ {ℓ} {A B : Type ℓ} {p : A ≡ B} → p ≡ refl · p
lUnitT {ℓ}{A}{B}{p} j i = lUnitT-filler p i1 j i
```

```
rCancelT : ∀ {ℓ} {A B : Type ℓ} (p : A ≡ B) → p · sym p ≡ refl
rCancelT {ℓ}{A}{B} p j i = rCancelT-filler p i1 j i

lCancelT : ∀ {ℓ} {A B : Type ℓ} (p : A ≡ B) → sym p · p ≡ refl
lCancelT {ℓ}{A}{B} p = rCancelT (sym p)
```

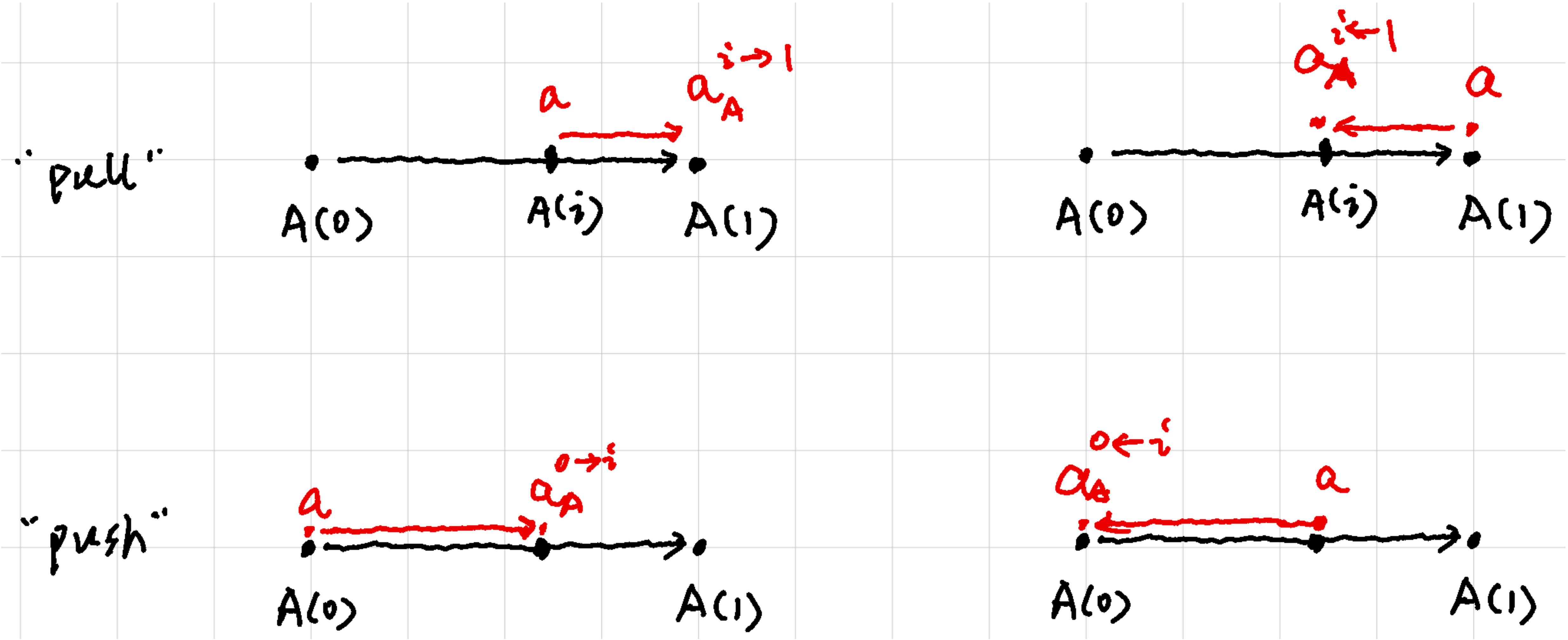
p is a path between types

GroupoidLawT.agda

```
assocT :  ∀ {ℓ} {A B C D : Type ℓ} {p : A ≡ B} {q : B ≡ C} {r : C ≡ D}
       → (p · q) · r ≡ p · (q · r)
assocT {ℓ}{A}{B}{C}{D}{p}{q}{r} j i = hcomp walls (p i)
  where
    walls : ∀ k → Partial (~ j ∨ j ∨ ~ i ∨ i) (Type ℓ)
    walls k (i = i0) = A
    walls k (i = i1) = (q · r) k
    walls k (j = i0) = β-filler p q r i k
    walls k (j = i1) = α-filler p q r i k
```



“CartesianKanOps” Model



Based on Week3

“Diagonal” Model

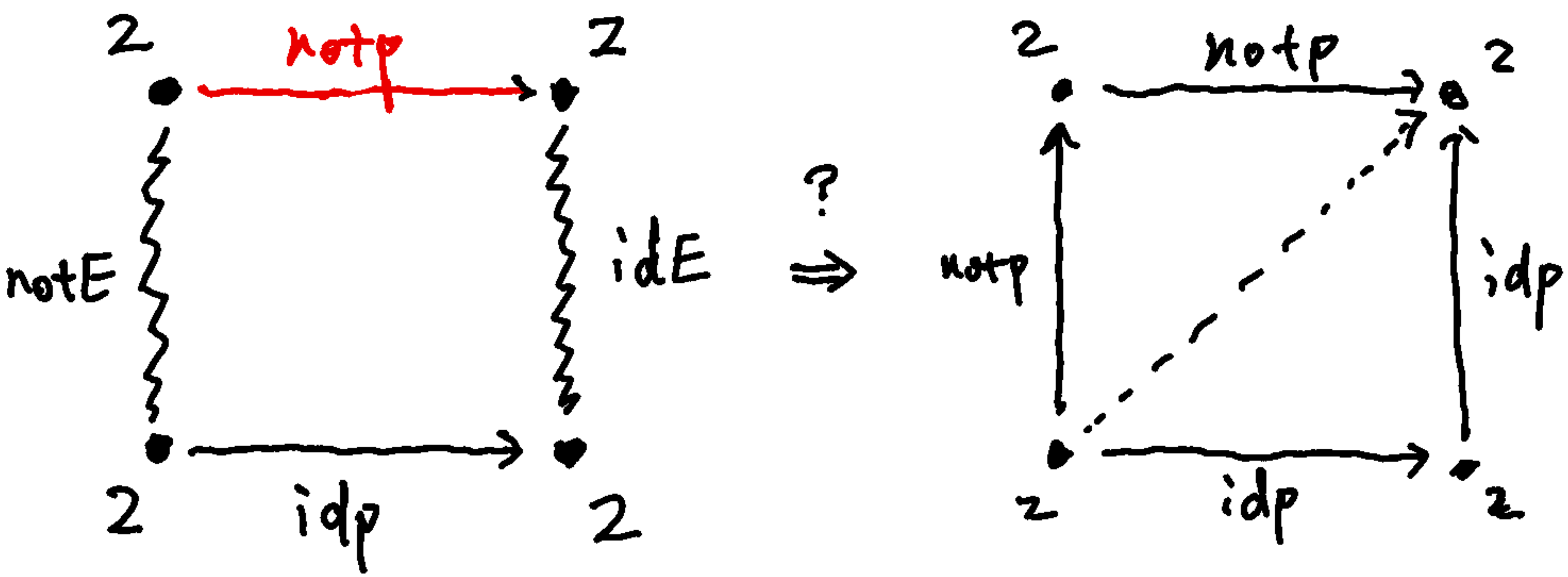
```
-- It's relatively easier to get the diagonal from a well-defined square
-- Square [left] [right] [bottom] [top]
diag-from-sq : (p q : Bool ≡ Bool)
             → Square p q q p → Bool ≡ Bool
diag-from-sq p q sq = λ i → sq i i
```

```
rem : (b : Bool) → not (not b) ≡ b
rem false = refl
rem true  = refl

notp : Bool ≡ Bool
notp = isoToPath (iso not not rem rem)

not-equiv : Bool ≃ Bool
not-equiv = isoToEquiv (iso not not rem rem)
```

```
notp' : Bool ≡ Bool
notp' i = Glue Bool walls
  where
    walls : Partial (~ i ∨ i) (Σ[ T ∈ Type ] (T ≃ Bool))
    walls (i = i0) = Bool , not-equiv
    walls (i = i1) = Bool , idEquiv Bool
```

“Diagonal” Model

How to get well defined cubes?



Questions

- `sqrt` related 2-combinators: more or less?
- semantics other than mapping to paths?
- other `sqrt` semantics model we can try within Cubical Agda?
- how can Bool type gets extended to quantum?
- measure, superposition, …