([Directed] Higher) Inductive Types in Bicubical Directed Type Theory

Matthew Z. Weaver and **Daniel R. Licata**

Princeton University

Wesleyan University

HoTT/UF Workshop
July 18th, 2021

What is Directed Type Theory?

- Directed type theory is an extension of homotopy type theory containing both invertible (regular) paths and directed paths (i.e. morphisms)
- Many other people have also studied directed type theory using a number of different approaches [Buchholtz & Weinberger, North, Nuyts, Riehl & Shulman]
- Our work extends from cubical type theory, allowing us to develop a constructive version of directed type theory
    - While we have developed a constructive foundation for directed type theory, much of the work I'll be talking about today is still work in progress

Let's see what using directed type theory could look like!

Let’s formalize the untyped lambda calculus

Standard

```
data Nat : Type where
  z   : Nat
  suc : Nat → Nat
```

```
Fin : Nat → Type
Fin z       = ⊥
Fin (suc Γ) = (Fin Γ) + ⊤
```

```
data Tm (Γ : Nat) : Type where
  var : Fin Γ → Tm Γ
  app : Tm Γ → Tm Γ → Tm Γ
  abs : Tm (suc Γ) → Tm Γ
```

Directed

$\mathtt{Nat}_{\leq}$ has unique compositions of morphisms

(i.e. $\mathtt{Nat}_{\leq}$ is a category)

```
data Nat≤ : USegal where
  z   : Nat≤
  suc : Nat≤ → Nat≤
  les : (Γ : Nat≤) → Hom Γ (suc Γ)
```

Let’s formalize the untyped lambda calculus

Standard

```
data Nat : Type where
  z   : Nat
  suc : Nat → Nat

Fin : Nat → Type
Fin z       = ⊥
Fin (suc Γ) = (Fin Γ) + ⊤

data Tm (Γ : Nat) : Type where
  var : Fin Γ → Tm Γ
  app : Tm Γ → Tm Γ → Tm Γ
  abs : Tm (suc Γ) → Tm Γ
```

Directed

```
data Nat≤ : USegal where
  z   : Nat≤
  suc : Nat≤ → Nat≤
  les : (Γ : Nat≤) → Hom Γ (suc Γ)

Fin : Nat≤ → UCov
Fin z       = ⊥
Fin (suc Γ) = (Fin Γ) + ⊤
Fin (les Γ) = ? : Hom_UCov (Fin Γ) (Fin (suc Γ))
```

Fin depends on the morphisms in $\mathrm{Nat}_{\leq}$ covariantly

Let’s formalize the untyped lambda calculus

Standard

```
data Nat : Type where
  z   : Nat
  suc : Nat → Nat

Fin : Nat → Type
Fin z       = ⊥
Fin (suc Γ) = (Fin Γ) + ⊤

data Tm (Γ : Nat) : Type where
  var : Fin Γ → Tm Γ
  app : Tm Γ → Tm Γ → Tm Γ
  abs : Tm (suc Γ) → Tm Γ
```

Directed

```
data Nat≤ : USegal where
  z   : Nat≤
  suc : Nat≤ → Nat≤
  les : (Γ : Nat≤) → Hom Γ (suc Γ)

Fin : Nat≤ → UCov
Fin z       = ⊥
Fin (suc Γ) = (Fin Γ) + ⊤
Fin (les Γ) = dua (? : Fin Γ → Fin (suc Γ))
```

Fin depends on the morphisms in $Nat_{\leq}$ covariantly

```
dua : {A B : UCov} → (A → B) → Hom A B
```

Let's formalize the untyped lambda calculus

Standard

```
data Nat : Type where
  z   : Nat
  suc : Nat → Nat
```

```
Fin : Nat → Type
Fin z       = ⊥
Fin (suc Γ) = (Fin Γ) + ⊤
```

```
data Tm (Γ : Nat) : Type where
  var : Fin Γ → Tm Γ
  app : Tm Γ → Tm Γ → Tm Γ
  abs : Tm (suc Γ) → Tm Γ
```

Directed

```
data Nat≤ : USegal where
  z   : Nat≤
  suc : Nat≤ → Nat≤
  les : (Γ : Nat≤) → Hom Γ (suc Γ)
```

```
Fin : Nat≤ → UCov
Fin z       = ⊥
Fin (suc Γ) = (Fin Γ) + ⊤
Fin (les Γ) = dua inl
```

Fin depends on the morphisms in $Nat_{\leq}$ covariantly

```
dua : {A B : UCov} → (A → B) → Hom A B
```

Let's formalize the untyped lambda calculus

Standard

```
data Nat : Type where
  z   : Nat
  suc : Nat → Nat


Fin : Nat → Type
Fin z       = ⊥
Fin (suc Γ) = (Fin Γ) + ⊤


data Tm (Γ : Nat) : Type where
  var : Fin Γ → Tm Γ
  app : Tm Γ → Tm Γ → Tm Γ
  abs : Tm (suc Γ) → Tm Γ
```

Directed

```
data Nat≤ : USegal where
  z   : Nat≤
  suc : Nat≤ → Nat≤
  les : (Γ : Nat≤) → Hom Γ (suc Γ)

Fin : Nat≤ → UCov
Fin z       = ⊥
Fin (suc Γ) = (Fin Γ) + ⊤
Fin (les Γ) = dua inl
```

Tm depends on the morphisms in $Nat_{\leq}$ covariantly

```
data Tm (Γ : Nat≤) : UCov where
  var : Fin Γ → Tm Γ
  app : Tm Γ → Tm Γ → Tm Γ
  abs : Tm (suc Γ) → Tm Γ
```

Let's formalize the untyped lambda calculus

Standard

Directed

```
data Tm (Γ : Nat) : Type where
  var : Fin Γ → Tm Γ
  app : Tm Γ → Tm Γ → Tm Γ
  abs : Tm (suc Γ) → Tm Γ
```

```
wk-Tm : ∀ Γ → Tm Γ → Tm (suc Γ)
wk-Tm Γ (var x) = var (inl x)
wk-Tm Γ (app t₁ t₂) = app (wk-Tm Γ t₁)
                          (wk-Tm Γ t₂)
wk-Tm Γ (abs t) = abs ??
```

Need to weaken under the bound variable

Let’s formalize the untyped lambda calculus

Standard **Directed**

```
Fin : Nat → Type
Fin z        = ⊥
Fin (suc Γ) = (Fin Γ) + ⊤

Loc : Nat → Type
Loc z = ⊥ + ⊤
Loc (suc Γ) = (Loc Γ) + ⊤

wk-Var : ∀ Γ → Loc Γ → Fin Γ → Fin (suc Γ)
wk-Var z l x = abort x
wk-Var (suc Γ) (inr l) x = inl x
wk-Var (suc Γ) (inl l) (inr x) = inr x
wk-Var (suc Γ) (inl l) (inl x) = inl (wk-Var Γ l x)
```

Let's formalize the untyped lambda calculus

Standard

Directed

```
data Tm (Γ : Nat) : Type where
  var : Fin Γ → Tm Γ
  app : Tm Γ → Tm Γ → Tm Γ
  abs : Tm (suc Γ) → Tm Γ

wk-Tm' : ∀ Γ → Loc Γ → Tm Γ → Tm (suc Γ)
wk-Tm' Γ l (var x) = var (wk-Var Γ l x)
wk-Tm' Γ l (app t₁ t₂) = app (wk-Tm' Γ l t₁)
                             (wk-Tm' Γ l t₂)
wk-Tm' Γ l (abs t) = abs (wk-Tm' (suc Γ) (inl l) t)
```

Let’s formalize the untyped lambda calculus

Standard

```
data Tm (Γ : Nat) : Type where
  var : Fin Γ → Tm Γ
  app : Tm Γ → Tm Γ → Tm Γ
  abs : Tm (suc Γ) → Tm Γ
```

```
wk-Tm' : ∀ Γ → Loc Γ → Tm Γ → Tm (suc Γ)
wk-Tm' Γ l (var x) = var (wk-Var Γ l x)
wk-Tm' Γ l (app t₁ t₂) = app (wk-Tm' Γ l t₁)
                             (wk-Tm' Γ l t₂)
wk-Tm' Γ l (abs t) = abs (wk-Tm' (suc Γ) (inl l) t)
```

```
wk-Tm : ∀ Γ → Tm Γ → Tm (suc Γ)
wk-Tm Γ t = wk-Tm' Γ (inr unit) t
```

Directed

```
data Nat≤ : USegal where
  z   : Nat≤
  suc : Nat≤ → Nat≤
  les : (Γ : Nat≤) → Hom Γ (suc Γ)
```

```
wk-Tm : ∀ Γ → Tm Γ → Tm (suc Γ)
wk-Tm Γ = ?
```

Let’s formalize the untyped lambda calculus

Standard

```
data Tm (Γ : Nat) : Type where
  var : Fin Γ → Tm Γ
  app : Tm Γ → Tm Γ → Tm Γ
  abs : Tm (suc Γ) → Tm Γ

wk-Tm' : ∀ Γ → Loc Γ → Tm Γ → Tm (suc Γ)
wk-Tm' Γ l (var x) = var (wk-Var Γ l x)
wk-Tm' Γ l (app t₁ t₂) = app (wk-Tm' Γ l t₁)
                             (wk-Tm' Γ l t₂)
wk-Tm' Γ l (abs t) = abs (wk-Tm' (suc Γ) (inl l) t)

wk-Tm : ∀ Γ → Tm Γ → Tm (suc Γ)
wk-Tm Γ t = wk-Tm' Γ (inr unit) t
```

Directed

```
data Nat≤ : USegal where
  z   : Nat≤
  suc : Nat≤ → Nat≤
  les : (Γ : Nat≤) → Hom Γ (suc Γ)

wk-Tm : ∀ Γ → Tm Γ → Tm (suc Γ)
wk-Tm Γ = dtransp Tm (? : Hom Γ (suc Γ))

dtransp : (A : C → UCov) → Hom_C c₁ c₂ → A c₁ → A c₂
```

Let’s formalize the untyped lambda calculus

Standard

```
data Tm (Γ : Nat) : Type where
  var : Fin Γ → Tm Γ
  app : Tm Γ → Tm Γ → Tm Γ
  abs : Tm (suc Γ) → Tm Γ

wk-Tm' : ∀ Γ → Loc Γ → Tm Γ → Tm (suc Γ)
wk-Tm' Γ l (var x) = var (wk-Var Γ l x)
wk-Tm' Γ l (app t₁ t₂) = app (wk-Tm' Γ l t₁)
                             (wk-Tm' Γ l t₂)
wk-Tm' Γ l (abs t) = abs (wk-Tm' (suc Γ) (inl l) t)

wk-Tm : ∀ Γ → Tm Γ → Tm (suc Γ)
wk-Tm Γ t = wk-Tm' Γ (inr unit) t
```

Directed

```
data Nat≤ : USegal where
  z   : Nat≤
  suc : Nat≤ → Nat≤
  les : (Γ : Nat≤) → Hom Γ (suc Γ)

wk-Tm : ∀ Γ → Tm Γ → Tm (suc Γ)
wk-Tm Γ = dtransp Tm (les Γ)
```

Let’s formalize the untyped lambda calculus

Directed

```
data Tm (Γ : Nat) : UCov where
  var : Fin Γ → Tm Γ
  app : Tm Γ → Tm Γ → Tm Γ
  abs : Tm (suc Γ) → Tm Γ

wk-Tm : ∀ Γ → Tm Γ → Tm (suc Γ)
wk-Tm Γ = dtransp Tm (les Γ)

wk-Tm-twice : ∀ Γ → Tm Γ → Tm (suc (suc Γ))
wk-Tm-twice Γ = dtransp Tm (les (suc Γ) ∘ les Γ)

wk-Tm-second : ∀ Γ → Tm (suc Γ) → Tm (suc (suc Γ))
wk-Tm-second Γ = dtransp Tm (dcong suc (les Γ))
```

```
Fin : Nat≤ → UCov
Fin z         = ⊥
Fin (suc Γ) = (Fin Γ) + ⊤
Fin (les Γ) = dua inl
```

We get all of this from having described how to weaken once at the outermost position!

And all these functions ***compute!***

From HoTT to Directed Type Theory

- Riehl-Shulman defines a type theory for ∞-categories with a model in bisimplicial sets
  1. Begin with HoTT
  2. Add Hom-types
  3. ∞-categories (Segal types) are described internally as predicates on types
  4. A predicate isCov(B : A → U) describes covariant discrete fibrations
  5. Cavallo, Riehl and Sattler have also (externally) defined the universe of covariant fibrations (the ∞-category of spaces and continuous functions) and shown it satisfies *Directed Univalence:* $\mathrm{Hom}_{U_{Cov}}\ A\ B \simeq A \to B$

From Bisimplicial to Bicubical Sets

- Can we make this constructive? Yes!
  1. Begin with Cubical Type Theory
  2. Use a second cubical interval to define Hom-types
  3. Use LOPS to define universes of Segal types and of covariant fibrations
  4. Construct directed univalence for the universe of covariant fibrations
     - ...or rather a sufficient subuniverse of the universe of covariant fibrations

Type Theory in Bicubical Sets

- As we are in bicubical sets, we have two interval objects:
  - We use the first interval to describe path structure

$$\frac{}{\mathbb{I} : \mathrm{Type}}$$

$$\frac{}{0_{\mathbb{I}} : \mathbb{I}} \qquad \frac{}{1_{\mathbb{I}} : \mathbb{I}}$$

  - The second interval to describes morphism structure

$$\frac{}{\mathbb{2} : \mathrm{Type}} \qquad \frac{x : \mathbb{2} \quad y : \mathbb{2}}{x \wedge y : \mathbb{2}} \qquad \frac{x : \mathbb{2} \quad y : \mathbb{2}}{x \vee y : \mathbb{2}}$$

$$\frac{}{0_{\mathbb{2}} : \mathbb{2}} \qquad \frac{}{1_{\mathbb{2}} : \mathbb{2}}$$

and equations...

  - We include connections for morphisms so we can define $x \leq y := x = x \wedge y$

Cofibrations

- We add cofibration propositions to describe boundaries of cubes

$$\frac{}{\text{isCof} : \Omega \to \Omega}$$

$$\text{Cof} := \Sigma\, \phi : \Omega\, .\, \text{isCof}\ \phi$$

Cof closed under $\_\wedge\_$, $\_\vee\_$, $\bot$, $\top$

$$\frac{x : \mathbb{I} \quad y : \mathbb{I}}{\_ : \text{isCof}\ (x = y)} \qquad \frac{x : \mathbb{2} \quad y : \mathbb{2}}{\_ : \text{isCof}\ (x = y)}$$

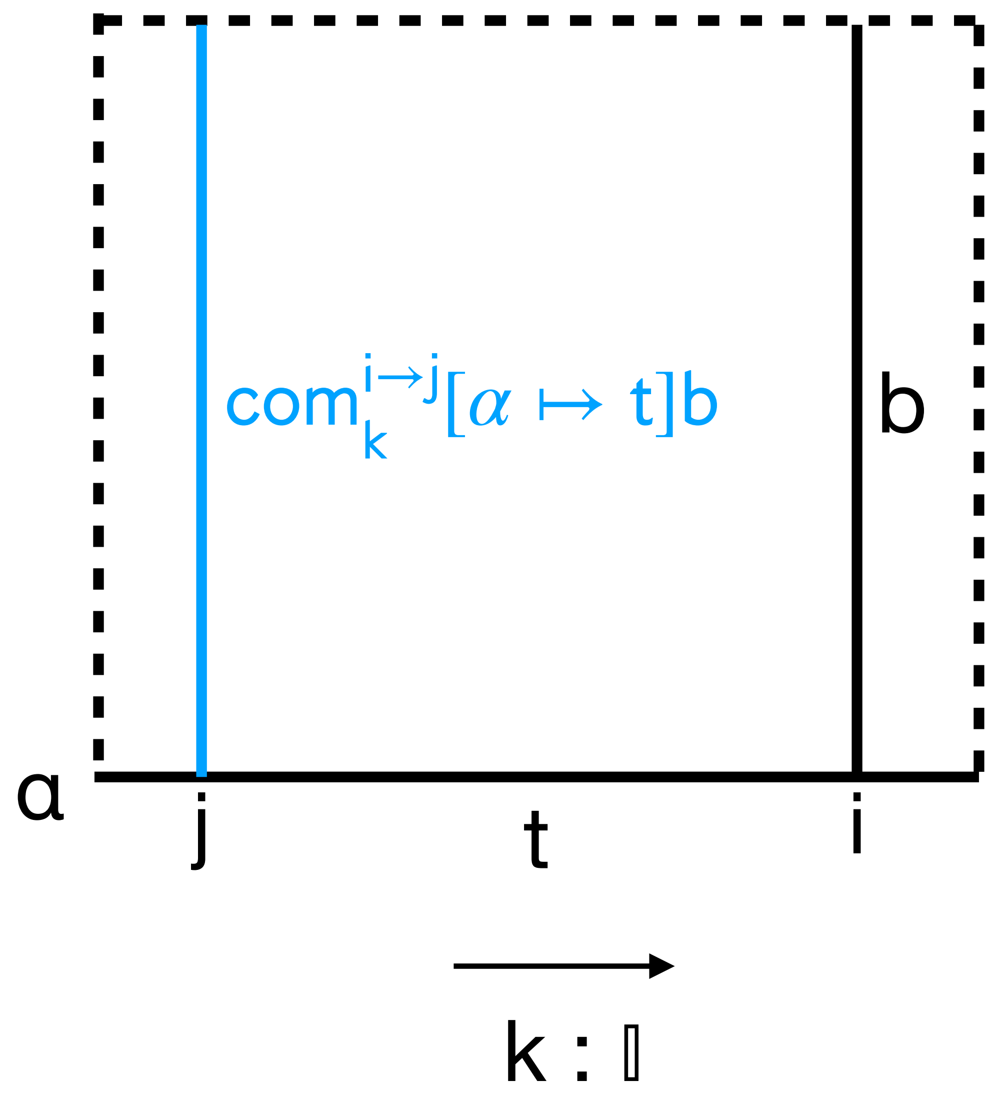
$$\frac{\phi : \mathbb{I} \to \text{Cof}}{\_ : \text{isCof}\ (\Pi\, x : \mathbb{I}\, .\, \phi\ x)} \qquad \frac{\phi : \mathbb{2} \to \text{Cof}}{\_ : \text{isCof}\ (\Pi\, x : \mathbb{2}\, .\, \phi\ x)}$$

Let's now classify well behaved types

Kan Types

$$
\begin{aligned}
&\mathrm{hasCom} : (\mathbb{I} \to U) \to U \\
&\mathrm{hasCom}\ A = \Pi\ i\ j : \mathbb{I}\ . \\
&\qquad\qquad \Pi\ \alpha : \mathrm{Cof}\ . \\
&\qquad\qquad \Pi\ t : (\Pi\ x : \mathbb{I}\ .\ \alpha \to A\ x) \\
&\qquad\qquad \Pi\ b : (A\ i)[\alpha \mapsto t\ i]\ . \\
&\qquad\qquad (A\ j)[\alpha \mapsto t\ j;\ i = j \mapsto b]
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{relCom} : (A : U) \to (A \to U) \to U \\
&\mathrm{relCom}\ A\ B = \Pi\ p : \mathbb{I} \to A\ . \\
&\qquad\qquad \mathrm{hasCom}\ (B \circ p)
\end{aligned}
$$

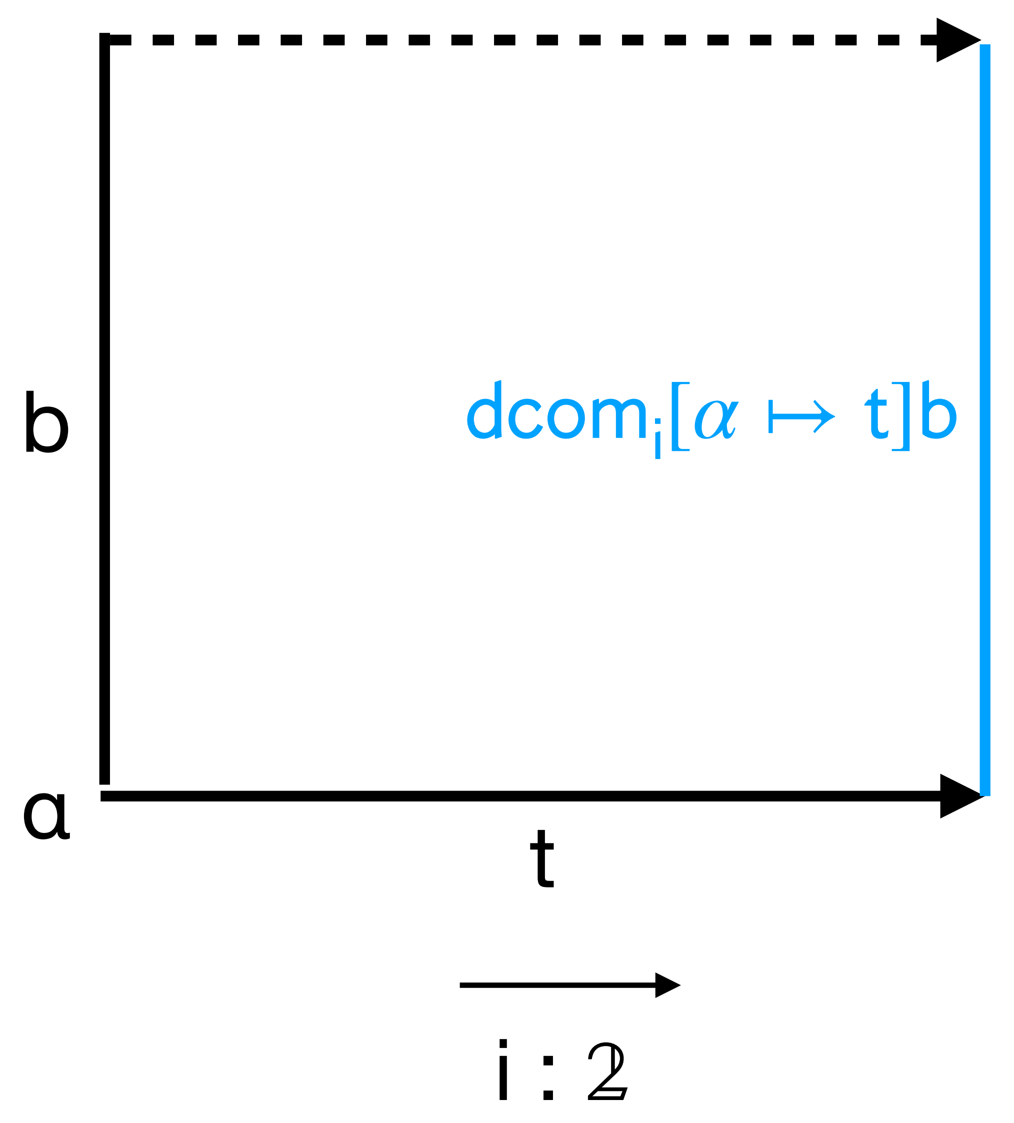


- If $B : A \to U$ satisfies relCom, the homotopical structure acts like that of spaces
- Note that this indicates nothing about the directed structure

Covariant Types

$$
\begin{aligned}
&\mathrm{hasCov} : (\mathbb{2} \to U) \to U \\
&\mathrm{hasCov}\ A = \Pi\ \alpha : \mathrm{Cof}\ . \\
&\qquad \Pi\ t : (\Pi\ x : \mathbb{2}\ .\ \alpha \to A\ x) \\
&\qquad \Pi\ b : (A\ \mathbb{0}_{\mathbb{2}})[\alpha \mapsto t\ \mathbb{0}_{\mathbb{2}}]\ . \\
&\qquad (A\ \mathbb{1}_{\mathbb{2}})[\alpha \mapsto t\ \mathbb{1}_{\mathbb{2}}]
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{relCov} : (A : U) \to (A \to U) \to U \\
&\mathrm{relCov}\ A\ B = \Pi\ p : \mathbb{2} \to A\ . \\
&\qquad \mathrm{hasCov}\ (B \circ p)
\end{aligned}
$$

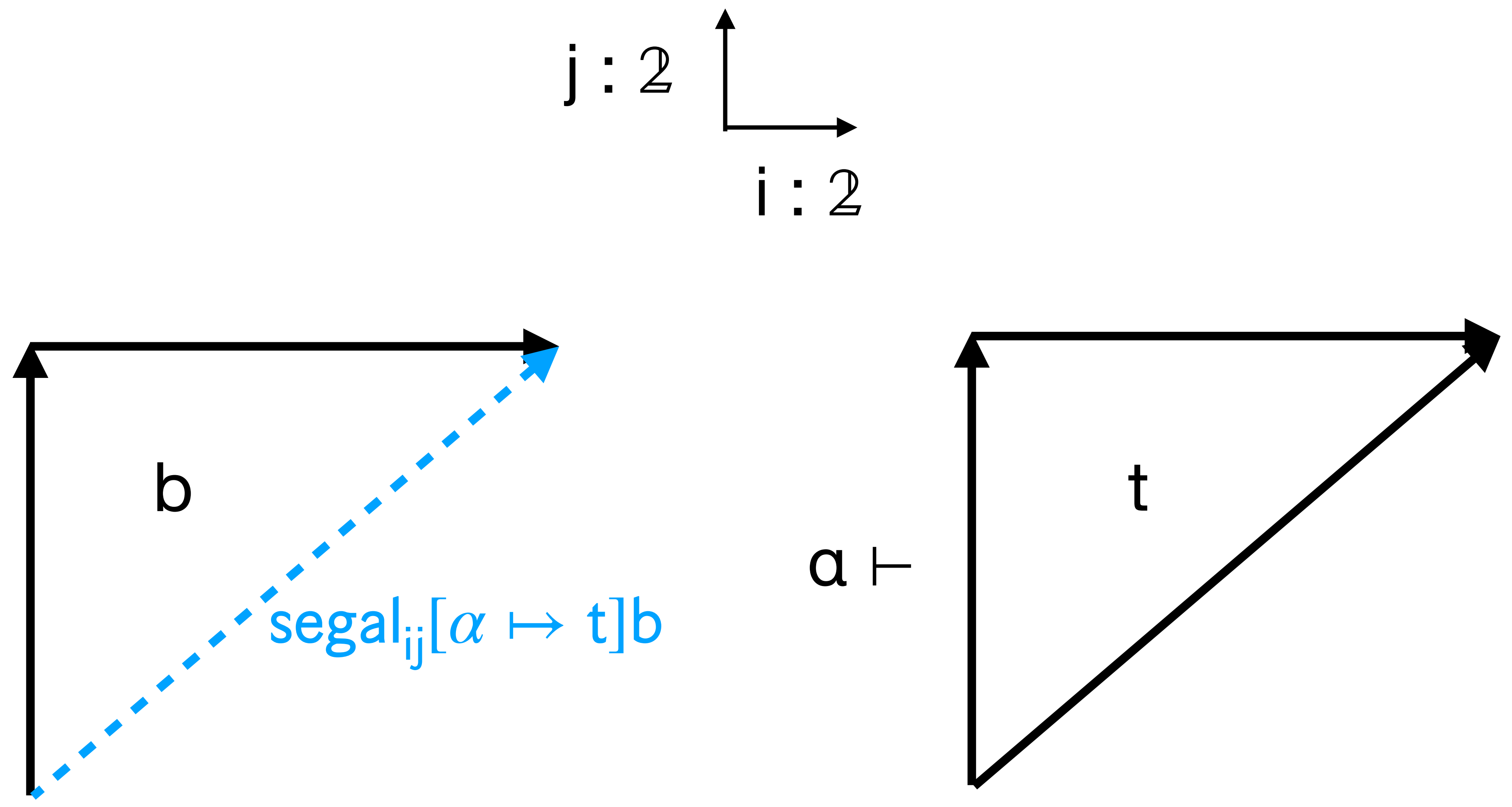


- If B : A → U satisfies relCov, the fibers of B are categorically discrete (i.e. are ∞-groupoids) and depend covariantly on the morphisms in A
- Thus, in the non-dependent case, hasCov (λ _ . A) means that A is categorically discrete

Segal Types

$\Delta := \Sigma\,(i\,,\,j) : \mathbb{2} \times \mathbb{2}\ .\ i \leq j$

$\Lambda := \Sigma\,(i\,,\,j) : \mathbb{2} \times \mathbb{2}\ .\ i = 0 \vee j = 1$



$$\begin{aligned}
&\mathrm{isSegal} : U \to U\\
&\mathrm{isSegal}\ A = \Pi\ \alpha : \mathrm{Cof}\ .\\
&\qquad \Pi\ t : (\Pi\ ij : \Delta\ .\ \alpha \to A)\\
&\qquad \Pi\ b : (\Pi\ ij : \Lambda.\ (A\ ij)[\alpha \mapsto t\ ij]\ .\\
&\qquad \Pi\ ij : \Delta.\ (A\ ij)[\alpha \mapsto t\ ij]
\end{aligned}$$

- If A : U satisfies isSegal, then A represents an ∞-category
- Note there is a dependent version called inner fibrations, studied in directed type theory by Buchholtz & Weinberger

Let's put these types in universes!

The LOPS Construction

- Consider a type J, and predicate on types $P : (J \to U) \to U$
- We then define $\mathrm{relP} : (\Gamma : U) \to (A : \Gamma \to U) \to U := (p : J \to \Gamma) \to P\ (A \circ p)$
- Licata, Orton, Pitts and Spitters define a constructive way to build a universe classifying type families satisfying relP assuming:
  - J is a tiny object
  - $\perp$ is a cofibration

UKan

- UKan is given by applying the LOPS construction to relCom
- Ignoring the directed structure, we get normal cubical type theory
- We can construct univalence for UKan

UCov

- relCov is itself Kan
- We define UCov by using the LOPS construction on relCov inside of UKan
- Types in UCov are thus both Kan and covariant
- We can construct univalence for UCov
- We can construct directed univalence for UCov

USegal

- isSegal is itself Kan
- We define USegal by restricting UKan to those types that are also Segal
  - (more generally, we could use LOPS with the inner fibration predicate)
- Types in USegal are thus both Kan and covariant
- We can construct univalence for USegal

So Many Universes!

- Let's recall our running example:

```
data Nat≤ : USegal where
  z   : Nat≤
  suc : Nat≤ → Nat≤
  les : (Γ : Nat≤) → Hom Γ (suc Γ)

Fin : Nat≤ → UCov
Fin z       = ⊥
Fin (suc Γ) = (Fin Γ) + ⊤
Fin (les Γ) = dua inl
```

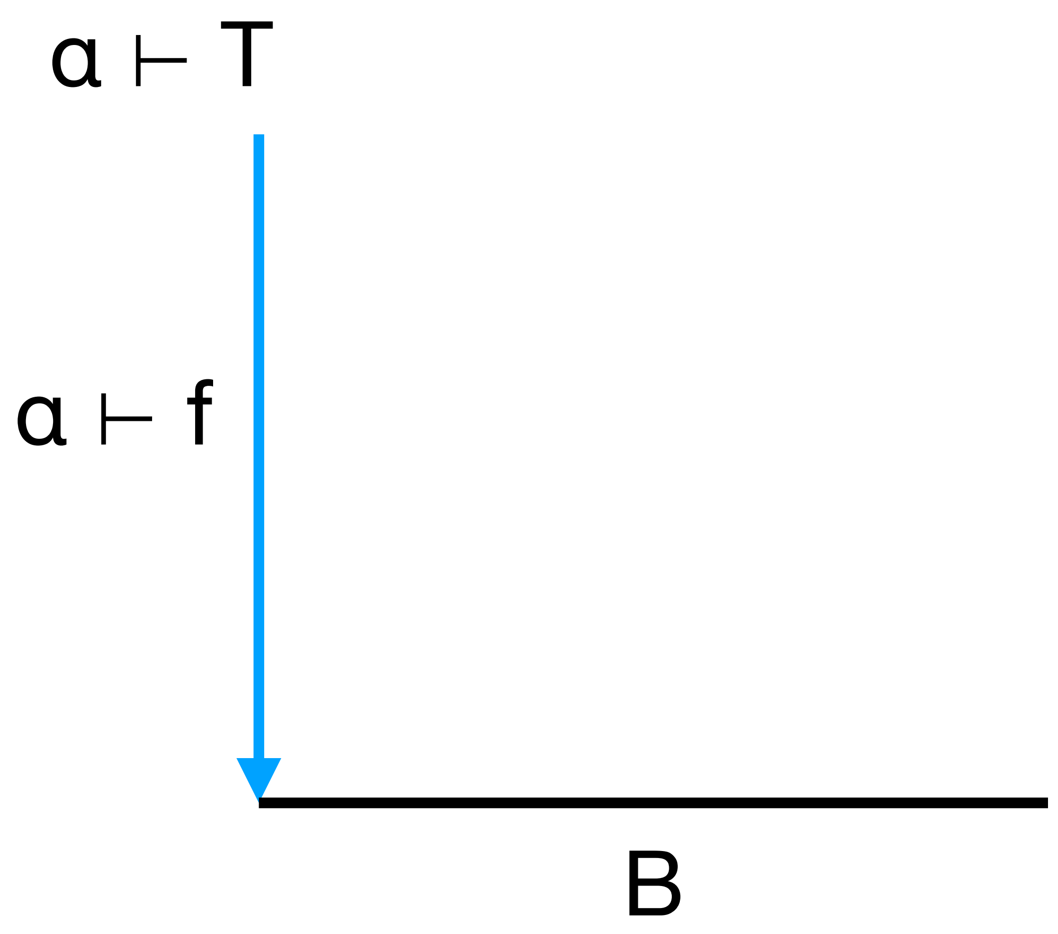
- Unlike in HoTT we need multiple universes classifying different classes of types
- As directed univalence is also fundamentally important to get this up and running, let's see how it works

Directed Univalence

- Recall directed univalence states that (A → B) is equivalent to $\mathrm{Hom}_{\mathrm{UCov}}$ A B for all A and B in UCov
- To construct directed univalence in UCov, we start by replicating the construction for regular univalence in cubical type theory
  - In particular, we use a directed version Glue types

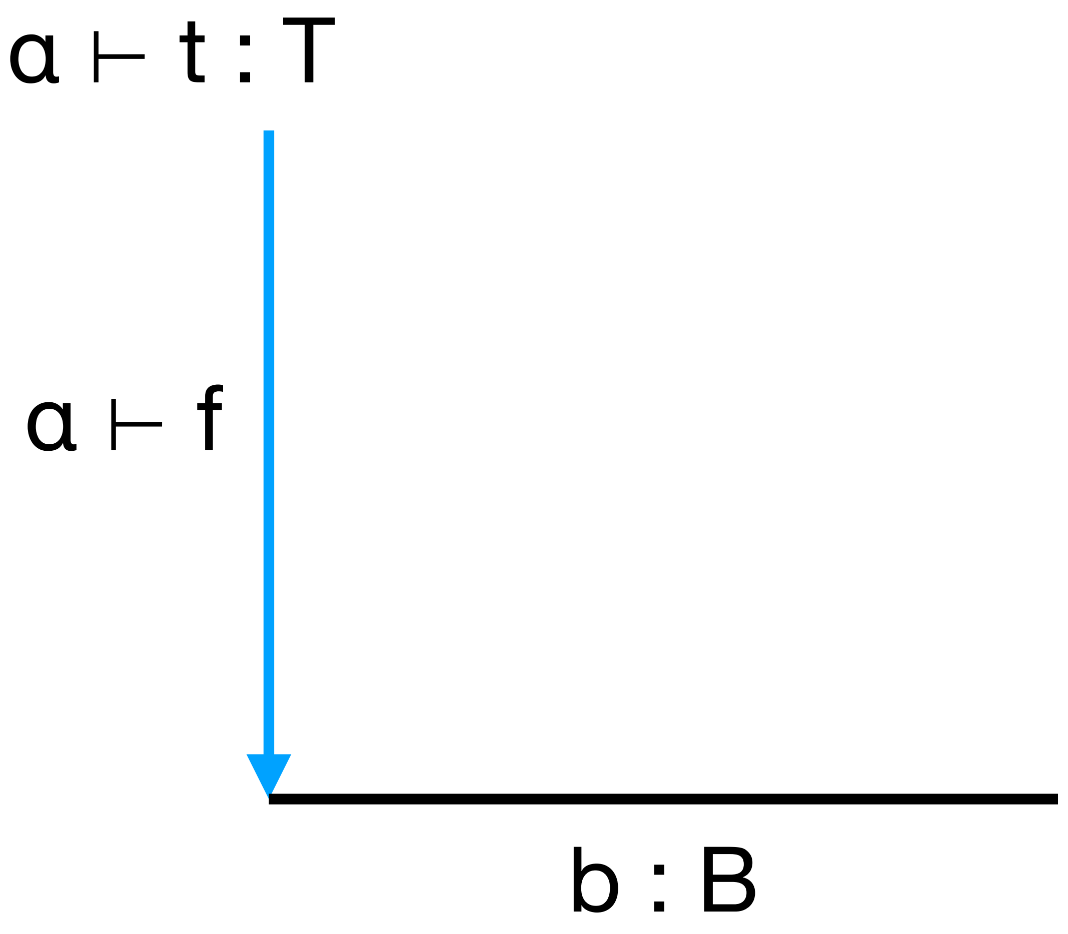
Directed Glue Types

Glue [α ↦ (T , f)] B :=



$$\alpha \vdash \mathrm{Glue}\ [\ \alpha \mapsto (T\ ,\ f)\ ]\ B \equiv T$$

glue t b :=



$$\frac{g : \mathrm{Glue}\ [\ \alpha \mapsto (T\ ,\ f)\ ]\ B}{\mathrm{unglue}\ g : B}$$

$$\alpha \vdash \mathrm{glue}\ t\ b \equiv t$$

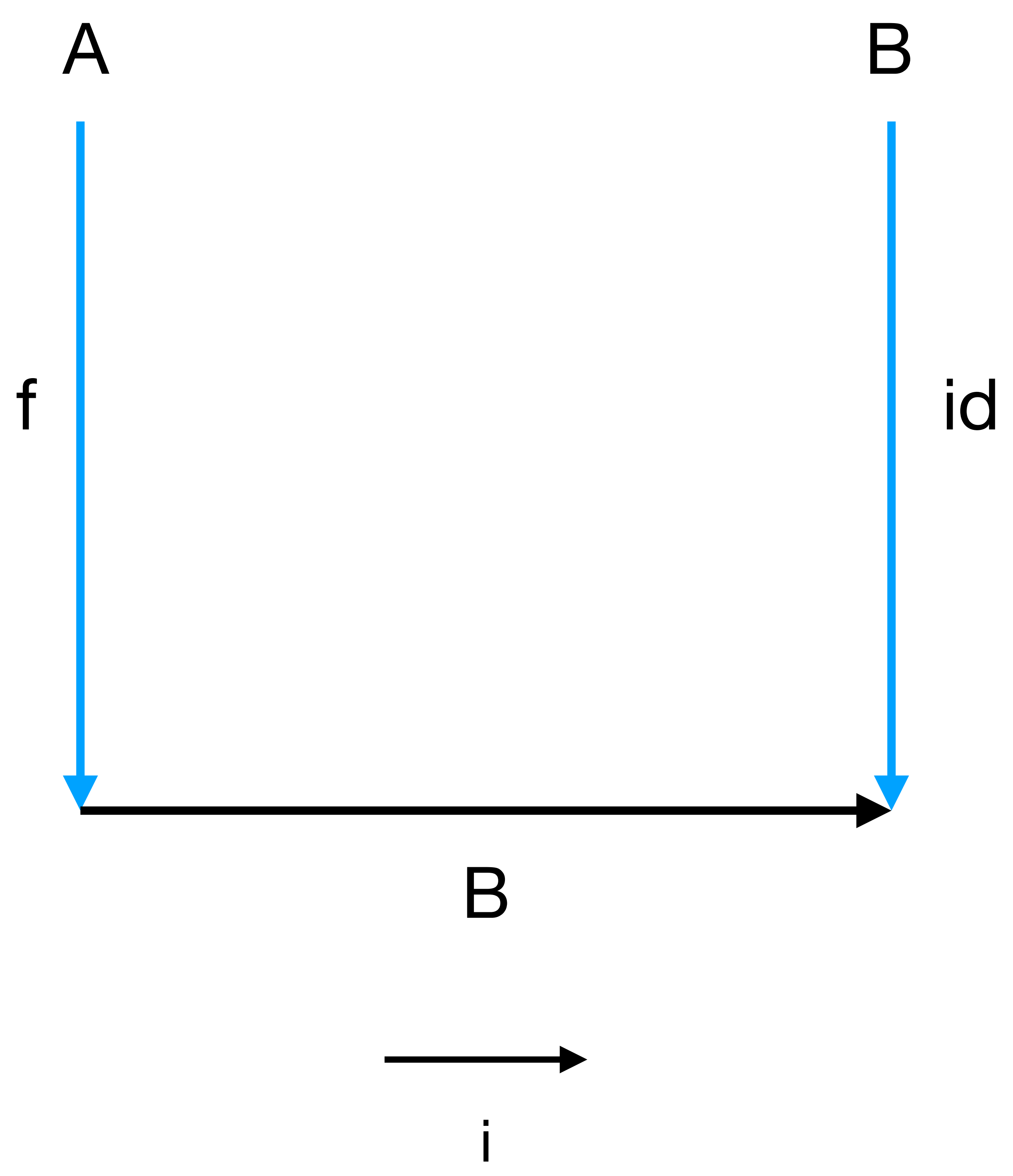
$$\alpha \vdash \mathrm{unglue}\ (\mathrm{glue}\ t\ b) \equiv f\ t$$

$$\mathrm{glue}\ g\ (\mathrm{unglue}\ g) \equiv g$$

(Almost) Directed Univalence

Given $f : A \to B$

$$\text{dua } i\ A\ B\ f := \lambda\, i\,.\ \text{Glue}\ [\ i = \mathbb{0}_{\mathbb{2}} \mapsto (A\ ,\ f)$$
$$,\ i = \mathbb{1}_{\mathbb{2}} \mapsto (B\ ,\ \text{id})\ ]\ B\ \ :\ \text{Hom}_{\text{UCov}}\ A\ B$$



(Almost) Directed Univalence

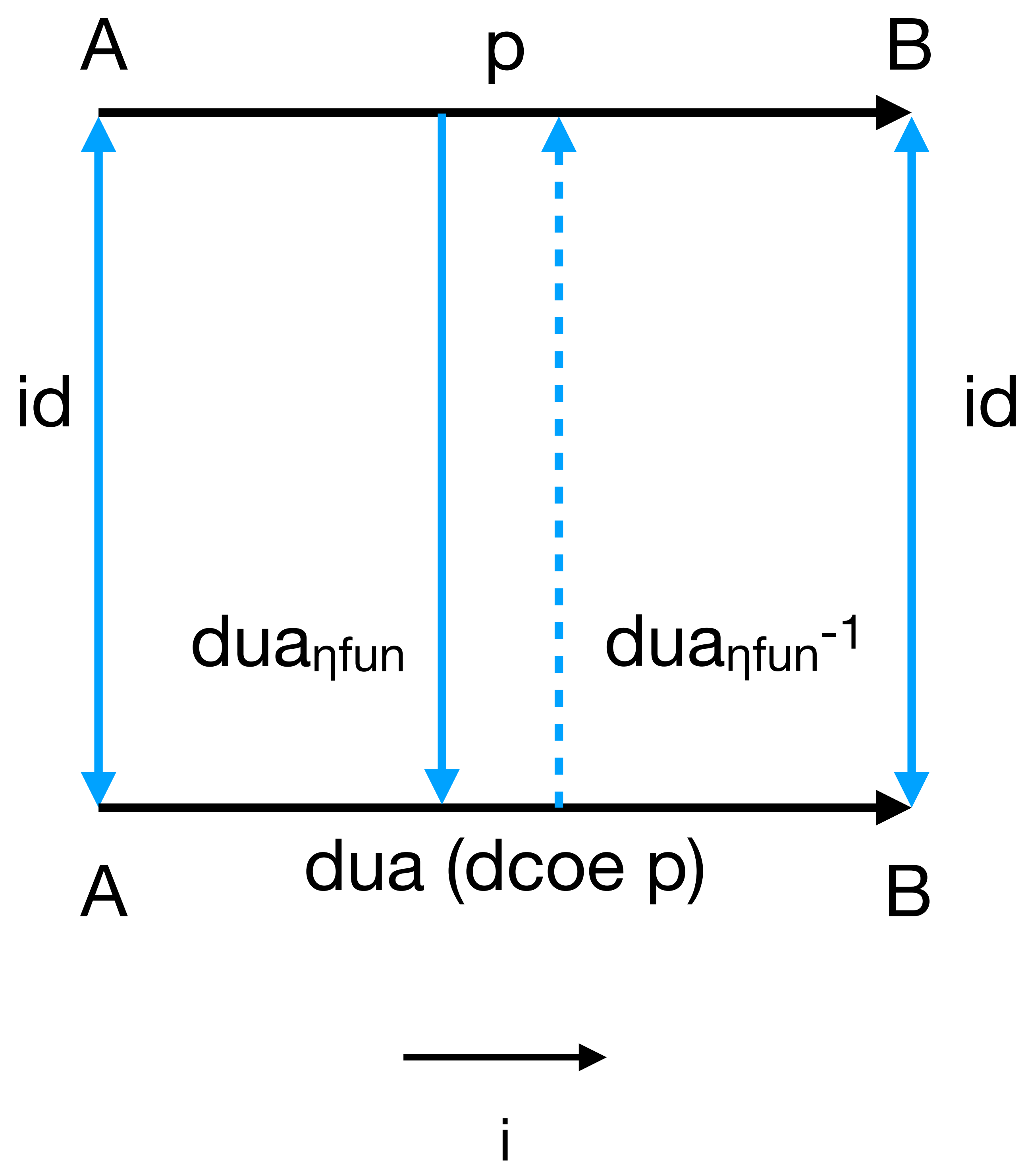
- dua is Kan + covariant, and thus lands in U_{Cov}
- U_{Cov} itself is Kan
- Path univalence holds in U_{Cov}
- These allow us to define the following for U_{Cov}:
  - dcoe : (Hom A B) → (A → B)
  - dua : (A → B) → Hom A B
  - dua_{β} : Π f : A → B . Path f (dcoe (dua f))
  - ~~dua_{η} : Π p : Hom A B . Path p (dua (dcoe p))~~

(Almost) Directed Univalence

- dua is Kan + covariant, and thus lands in U_{Cov}
- U_{Cov} itself is Kan
- Path univalence holds in U_{Cov}
- These allow us to define the following for U_{Cov}:
  - dcoe : (Hom A B) → (A → B)
  - dua : (A → B) → Hom A B
  - dua_{β} : Π f : A → B . Path f (dcoe (dua f))
  - $dua_{\eta fun}$: Π p : Hom A B . Π i : $\mathbb{2}$. p i → (dua (dcoe p)) i

(Almost) Directed Univalence

- We're thus left with the following picture:



- To complete directed univalence, we need dua$_{\eta fun}$$^{-1}$

Finishing Directed Univalence

- To show duaηfun is a homotopy equivalence, we restrict UCov to a useful subuniverse
  - Based on sheaf models of type theory by Coquand, Ruch and Sattler
  - This subuniverse is closed under the same type formers
  - Homotopy equivalences for bicubical sets in this universe are level-wise homotopy equivalences of cubical sets
  - Using this property we complete the proof of directed univalence (using the same technique as Cavallo, Riehl and Sattler did for the bisimplicial model)

Cubical HITs

- To warm up, let's review how HITs work in cubical type theory [Coquand, Huber & Mörtberg; Cavallo & Harper]

S1 in Cubical Type Theory

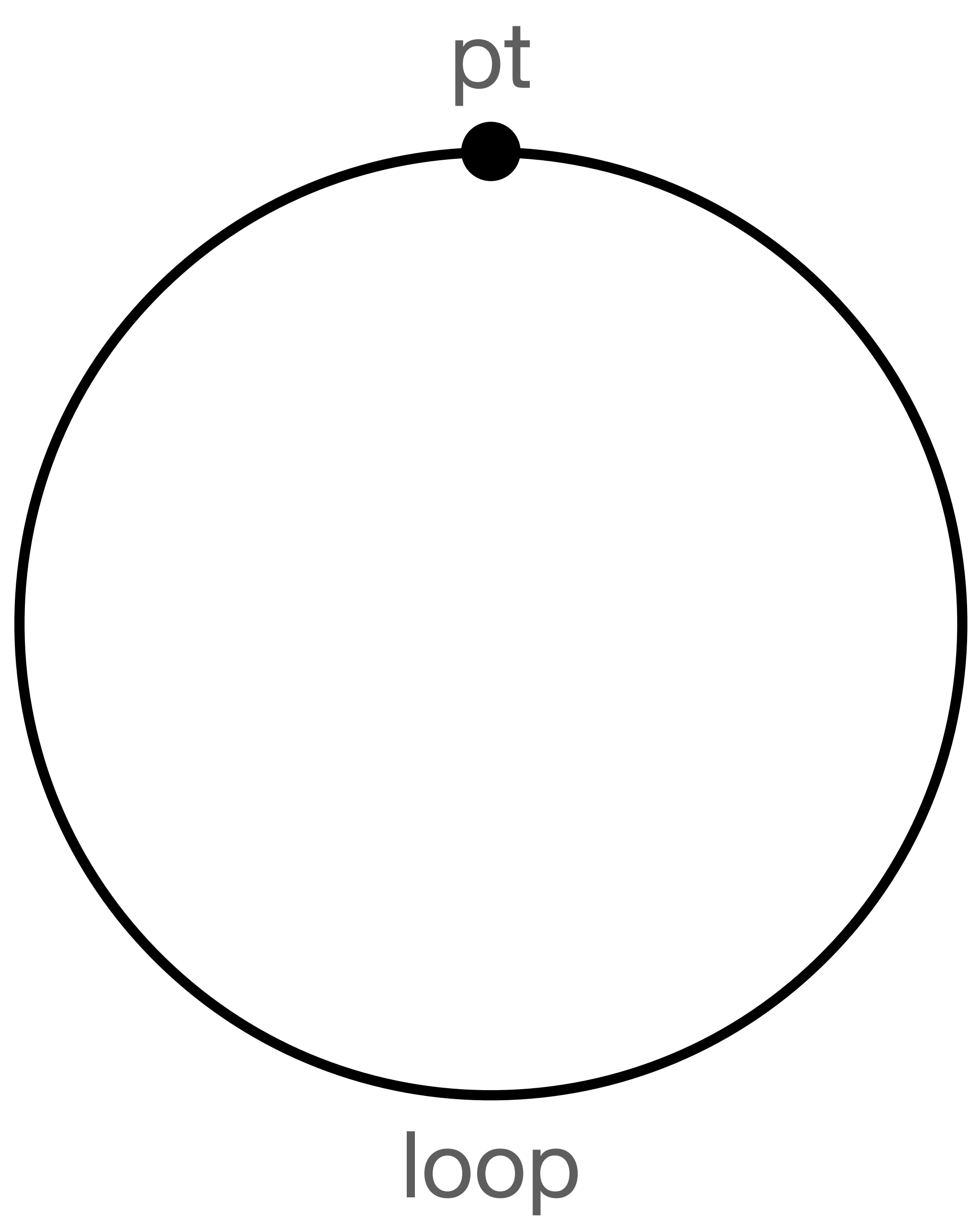
- Consider the circle, S1:
- We define it as the (homotopy) initial algebra generated by one point, and the loop connecting the point to itself
- Being initial, we expect to be able to map out of it by induction:

$$\frac{A : S1 \to \mathrm{UKan} \quad cp : A\ pt \quad cl : \mathrm{PathO}_{A \circ loop}\ p}{x : S1 \vdash \mathrm{S1elim}\ A\ cp\ cl\ x : A\ x}$$

- We also want it to be Kan/fibrant
    - To enforce this, we freely add solutions to the Kan filling problem

```
data S1 : UKan where
  pt   : S1
  loop : Path pt pt

  S1com : hasHCom S1
```



S1 in Cubical Type Theory

- How do we maintain the elimination principle while also including the additional Kan generator?
  - We only eliminate into Kan types, mapping the formal composition operator to the composition structure of the motive/codomain
- In the recursive case for A : UKan

$$\text{S1elim}\ A\ p\ l\ (\text{hcom}^{i \mapsto j}[\alpha \mapsto t]\ b) \equiv \text{hcom}^{i \mapsto j}[\alpha \mapsto x.\text{S1elim}\ A\ p\ l\ (t\ x)]\ (\text{S1elim}\ A\ p\ l\ b)$$

$\mathrm{Nat}_{\leq}$ in Bicubical Type Theory

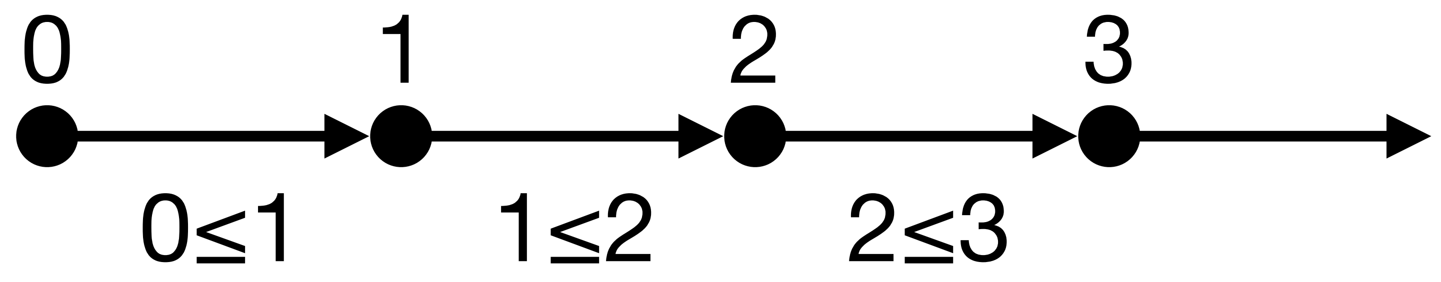
- Consider the category $\mathrm{Nat}_{\leq}$:
- We define it as the homotopy/Segal initial algebra generated by zero, the successor constructor, and the inequality morphism
- Being initial, we expect to be able to map out of it by induction:

$$\frac{A : \mathrm{Nat}_{\leq} \to \mathrm{USegal} \quad cz : A\ z \quad n : \mathrm{Nat}_{\leq} \vdash cs : A\ n \to A\ (suc\ n) \qquad n : \mathrm{Nat}_{\leq}, x : A\ n \vdash cl : \mathrm{PathO}_{A\ \circ\ (les\ n)}\ x\ (cs\ x)}{n : \mathrm{Nat}_{\leq} \vdash \mathrm{Nat}_{\leq}\mathrm{elim}\ A\ cz\ cs\ cl\ n : A\ n}$$

- We also want it to be Kan and Segal
  - To enforce this, we freely add solutions to the Kan and Segal filling problems

```
data Nat≤ : USegal where
  z   : Nat≤
  suc : Nat≤ → Nat≤
  les : (Γ : Nat≤) → Hom Γ (suc Γ)

  Nat≤com   : hasHCom Nat≤
  Nat≤Segal : isSegal Nat≤
```



$\mathrm{Nat}_{\leq}$ in Bicubical Type Theory

- How do we maintain the elimination principle while also including the additional Kan and Segal generators?
  - We only eliminate into types that are both Kan and Segal, mapping the formal filling operators to those of the motive/codomain
- In the recursive case for A : USegal

$$\mathrm{Nat}_{\leq}\mathrm{elim}\ A\ cz\ cs\ cl\ (\mathrm{segal}[\alpha \mapsto t]\ b) \equiv$$
$$\mathrm{segal}[\alpha \mapsto xy.\mathrm{Nat}_{\leq}\mathrm{elim}\ A\ cz\ cs\ cl\ (t\ xy)]\ (\mathrm{Nat}_{\leq}\mathrm{elim}\ A\ cz\ cs\ cl\ b)$$

How does Fin work?

- Recall we defined the following:

```
data Nat≤ : USegal where
  z   : Nat≤
  suc : Nat≤ → Nat≤
  les : (Γ : Nat≤) → Hom Γ (suc Γ)

Fin : Nat≤ → UCov
Fin z       = ⊥
Fin (suc Γ) = (Fin Γ) + ⊤
Fin (les Γ) = dua inl
```

- Let's look at how the function `wk-Var := dtransp Fin (les n)` computes

How does Fin work?

```
wk-Var Γ x := dtransp Fin (les Γ) x
  ⇒
dcom0↦1 (λi.Fin (les Γ i)) [⊥ ↦ ()] x
  ⇒
dcom0↦1 (dua inl) [⊥ ↦ ()] x
  ⇒
dcom0↦1 (λi.Glue[i=0 ↦ (Fin n, inl), i=1 ↦ (Fin (suc n), id)] (Fin (suc n)))[⊥ ↦ ()] x
  ⇒
glue[1=0 ↦ ...,
     1=1 ↦ dcom0↦1 (λi.Fin (suc n)) [⊥ ↦ ()] (unglue x)]
    (dcom0↦1 (λi.Fin (suc n)) [...] (unglue x))
  ⇒
dcom0↦1 (λi.Fin (suc n)) [⊥ ↦ ()] (unglue x)
  ⇒
dcom0↦1 (λi.Fin (suc n)) [⊥ ↦ ()] (inl x)
  ⇒
inl x
```

General DHITs

- Still a work in progress!
- We expect it should go similarly to cubical higher inductive types, but instead by adding both formal Kan composition and formal Segal composition

HITs and Universe Challenges

```
data S1 : UKan where
  pt   : S1
  loop : Path pt pt

  S1com : hasHCom S1
```

- Consider S1 in directed type theory
- In particular, notice what happens with the freely added compositions:
- As our cofibrations now let us talk about directed interval variables, S1 now has nontrivial homomorphisms!

$$\text{e.g. } \lambda\, i : \mathbb{2}\, .\mathrm{hcom}^{0\to 1}[i = 0 \mapsto pt, i = 1 \mapsto loop]pt : \mathrm{Hom}\ pt\ pt$$

- S1 should be categorically discreet (and thus land in UCov), but proving so is nontrivial