Todo list

Acknowledgements

Introduction

In type theory, the basic object of study is the type. Sitting at the lowest level, we can specify types whose *points* are freely generated by a list of constructors - these types are aptly named *inductive types.* From these inductive types, we can form sections to come to the notion of dependent products, or we can ask what type they live in to arrive at universes. We can also extend the notion of the inductive types themselves, allowing constructors to specify *paths* in the type, not just points to get *higher inductive types.* In a directed setting, the notion of a *directed* higher inductive type should correspond to the notion of a *category* (with some extra work). That is, certain directed higher inductive definitions of types satisfy the *Segal* property.

Contents

1. MLTT

2. HoTT

3. sHoTT

3.1 Three-layer type theory with shapes

3.2 Extension types

3.3 Segal condition

With our three-layer type theory and extension types, we can start to define familiar category theoretic notions. We start by defining important types which depend on shapes.

Definition 3.1. For a type A and points $x, y : A$ which induce a term $[x, y]$[1] in context $\partial\Delta^1$, we write

$$\hom_A(x, y) :\equiv \left\langle \Delta^1 \to A \middle|^{\partial\Delta^1}_{[x,y]} \right\rangle$$

and say that an element of $\hom_A(x, y)$ is a **morphism** from x to y.

Definition 3.2. For points $x, y, z : A$, and morphisms $f : \hom_A(x, y)$, $g : \hom_A(y, z)$, and $h : \hom_A(x, z)$ which induces a term $[x, y, z, f, g, h]$[2] in context $\partial\Delta^2$, we write

$$\hom^2_A\left(\begin{tikzcd} & y \arrow[dr, dash, "f"] & \\ x \arrow[ur, dash, "f"] \arrow[rr, dash, "h"'] & & z \end{tikzcd}\right) :\equiv \left\langle \Delta^2 \to A \middle|^{\partial\Delta^2}_{[x,y,z,f,g,h]} \right\rangle$$

or, in an abbreviated fashion,

$$\hom^2_A(f, g; h)$$

and say that an element of $\hom^2_A(f, g; h)$ is a 2-**morphism**.

Remark. We will mostly write a 2-morphism as

Not only do functions act functorially on paths, they also act functorially on morphisms and higher morphisms.

Lemma 3.1. *(Morphism application) Let A, B be types and $f : A \to B$. Then, given $x, y : A$, there are functions*

$$\mathsf{map}^{\to}_f : \hom_A(x, y) \to \hom_A(fx, fy)$$

[1]Spelled out, this is a term that depends on $t : \partial\Delta^1$ that evaluates to x when $t \equiv 0$ and y when $t \equiv 1$.

[2]Similarly, this is a term that depends on $\langle t, s\rangle : \partial\Delta^2$ that evaluates to:

- x when $\langle t, s\rangle \equiv \langle 0, 0\rangle$
- y when $\langle t, s\rangle \equiv \langle 1, 0\rangle$
- z when $\langle t, s\rangle \equiv \langle 1, 1\rangle$
- f when $\langle t, s\rangle \equiv \langle t, 0\rangle$
- g when $\langle t, s\rangle \equiv \langle 1, s\rangle$
- h when $\langle t, s\rangle \equiv \langle t, t\rangle$

In section 2, we saw that composition of paths was a function. With morphisms, we don't get the same luxury a priori. Instead, we can ask for types whose morphisms do have composites. So, composition of paths is a parametrically polymorphic function while composition of morphisms is a type class polymorphic function. That leads us to the following definition.

Definition 3.3. Let A be a type, $x, y, z : A$ and $f : \mathrm{hom}_A(x, y)$ and $g : \mathrm{hom}_A(y, z)$. We say A is **Segal** if the type

$$\sum_{h:\mathrm{hom}_A(x,z)} \mathrm{hom}^2_A(f, g; h)$$

is contractible. The unique inhabitant is denoted by $(g \circ f, \mathrm{comp}_{g,f})$.

In other words two composable morphisms are enough to specify a 2-morphism in a Segal type. That leads us to the following important alternative characterization of Segal-ness.

Lemma 3.2. *A type A is Segal if and only if the restriction map*

$$(\Delta^2 \to A) \to (\Lambda^2_1 \to A)$$

is an equivalence.

Remark. In the proof of (lemma reference), c.f proves along the way that there is an equivalence

$$\sum_{h:\mathrm{hom}_A(x,z)} \mathrm{hom}^2_A\left(\begin{tikzcd} & y \arrow[dr, dash, "g"] & \\ x \arrow[ur, dash, "f"] \arrow[rr, dash, "h"'] & & z \end{tikzcd}\right) \simeq \left\langle \Delta^2 \to A \middle|^{\Lambda^2_1}_{[x,y,z,f,g]} \right\rangle. \qquad (1)$$

The same argument applies to form equivalences

$$\sum_{f:\mathrm{hom}_A(x,y)} \mathrm{hom}^2_A\left(\begin{tikzcd} & y \arrow[dr, dash, "g"] & \\ x \arrow[ur, dash, "f"] \arrow[rr, dash, "h"'] & & z \end{tikzcd}\right) \simeq \left\langle \Delta^2 \to A \middle|^{\Lambda^2_2}_{[x,y,z,g,h]} \right\rangle \qquad (2)$$

and

$$\sum_{g:\mathrm{hom}_A(y,z)} \mathrm{hom}^2_A\left(\begin{tikzcd} & y \arrow[dr, dash, "g"] & \\ x \arrow[ur, dash, "f"] \arrow[rr, dash, "h"'] & & z \end{tikzcd}\right) \simeq \left\langle \Delta^2 \to A \middle|^{\Lambda^2_0}_{[x,y,z,f,h]} \right\rangle. \qquad (3)$$

Paths between morphisms also correspond two 2-morphisms:

Lemma 3.3. *For any $f, g : \mathrm{hom}_A(x, y)$ in a Segal type A, the natural map*

$$f = g \to \mathrm{hom}^2_A\left(\begin{tikzcd} & y \arrow[dr, dash, "f"] & \\ x \arrow[ur, dash, "\mathrm{idhom}_x"] \arrow[rr, dash, "g"'] & & z \end{tikzcd}\right)$$

is an equivalence.

Lemma 3.4. *For any $f : \hom_A(x, y)$, $g : \hom_A(y, z)$, and $h : \hom_A(x, z)$ in a Segal type A, the natural map*

$$g \cdot f = h \to \hom^2_A \left(\begin{tikzcd} & y \arrow[dr, dash, "g"] & \\ x \arrow[ur, dash, "f"] \arrow[rr, dash, "h"'] & & z \end{tikzcd} \right)$$

is an equivalence.

3.4 Discrete types

3.5 Covariant fibrations

In section 2, we saw that all type families were fibrations and what that means specifically in HoTT. Of course in sHoTT type families are also fibrations, with respect to *paths* that is. The question of which type families are fibrations with respect to *morphisms* and *higher morphisms*, is not as trivial in sHoTT. In this section, we define and characterize type families that lift morphisms. Before we can define such type families, we must first make precise what it means to be a "morphism in the total space".

Definition 3.4. Given a type A, a type family $C : A \to \mathcal{U}$, points $x, y : A$, a morphism $f : \hom_A(x, y)$ and points $u : C(x)$ and $v : C(y)$, the type of **dependent morphisms over** f **from** u to v is written as

$$\hom_{C(f)}(u, v) :\equiv \left\langle \textstyle\prod_{t:\Delta^1} C(f(t)) \Big|^{\partial\Delta^1}_{[u,v]} \right\rangle .$$

The dependent morphism represents the intuitive notion of "morphism in the total space". Alternatively, we can observe that a type family $C : A \to \mathcal{U}$ associates morphisms $f : \hom_A(x, y)$ to spans

$$\begin{tikzcd} & \sum_{(u:C(x))} \sum_{(v:C(y))} \hom_{C(f)}(u, v) \arrow[dl, "\mathrm{dom}"'] \arrow[dr, "\mathrm{cod}"] & \\ C(x) & & C(y) \end{tikzcd}$$

We do not know a priori that C depends functorially on A. We cannot even assert that *spans* depend functorially on A either, given

Definition 3.5. Given a type A and

$$x, y, z : A$$
$$f : \hom_A(x, y) \qquad g : \hom_A(y, z) \qquad h : \hom_A(x, z)$$
$$t : \hom^2_A \left(\begin{tikzcd} & y \arrow[dr, dash, "g"] & \\ x \arrow[ur, dash, "f"] \arrow[rr, dash, "h"'] & & z \end{tikzcd} \right)$$
$$u : C(x) \qquad v : C(y) \qquad w : C(z)$$
$$p : \hom_{C(f)}(u, v) \qquad q : \hom_{C(g)}(v, w) \qquad r : \hom_{C(h)}(u, w)$$

we define

$$\hom^2_{C(t)}\left(\begin{tikzcd}[ampersand replacement=\&] \& v \arrow[dr, "q"] \& \\ u \arrow[ur, "p"] \arrow[rr, "r"'] \& \& w\end{tikzcd}\right) :\equiv \left\langle \prod_{s:\Delta^2} C(t(s)) \Big|^{\partial\Delta^2}_{[u,v,w,p,q,r]} \right\rangle$$

and say that a morphism in $\hom^2_{C(t)}\left(\begin{tikzcd}[ampersand replacement=\&] \& v \arrow[dr, "q"] \& \\ u \arrow[ur, "p"] \arrow[rr, "r"'] \& \& w\end{tikzcd}\right)$ is a **dependent 2-simplex over** t.

In the case that $g \circ f = h$, we cannot necessarily conclude that the span associated to $C(h)$ is equal to the composites of the spans $C(g)$ and $C(f)$. These issues have easy workarounds, we can restrict ourselves to type families that act as fibrations with respect to morphisms, or alternatively functorial with respects to the fibers of C.

Definition 3.6. A type family is **covariant** if for every $f : \hom_A(x, y)$ and $u : C(x)$, the type

$$\sum_{v:C(y)} \hom_{C(f)}(u, v)$$

is contractible. Dually[3], C is **contravariant** if for every $f : \hom_A(x, y)$ and $v : C(y)$, the type

$$\sum_{v:C(x)} \hom_{C(f)}(u, v)$$

is contractible.

We make our first extension to sHoTT by assuming that there is a type that acts as a classifier for covariant fibrations.

Axiom. *There is a a Segal type $\mathcal{U}_{cov}$ such that for any type or shape A, type families $C : A \to \mathcal{U}_{cov}$ are covariant fibrations.*

Remark. Discrete families are members of $\mathcal{U}_{\text{cov}}$

We can also show that definition of a covariant type family really does correspond to the familiar notion of a fibration in topology:

Lemma 3.5. *A type family $C : A \to \mathcal{U}$ is covariant if and only if for all morphisms $f : \hom_A(x, y)$ and points $u : C(x)$ the type $\left\langle \prod_{t:\Delta^1} C(f(t)) \big|^0_u \right\rangle$ is contractible; there is a unique lifting of f that starts at u.*

Like in category theory, we can make extensive use of the covariance of the "representable" $\lambda x. \hom_A(a, x)$. To start, we see that the covariant of the representable allows us to characterize Segal types.

Proposition 3.6. *Given a type A and a point $a : A$, then*

$$\lambda x. \hom_A(a, x) : A \to \mathcal{U}_{\text{cov}}$$

if and only if A is Segal.

[3]We will not explicitly state dual theorems and definitions when they are obvious from context.

We also enjoy some closure properties with covariant type families: mapping into a covariant family is covariant and mapping out of a covariant family *into a discrete tpye* is contravariant.

Lemma 3.7 (RS17 Prop 8.30)**.** *For any types C, A, and covariant family $B : C \to \mathcal{U}$, the type family*

$$\lambda(x : C).\, A \to B(x) : C \to \mathcal{U}$$

is also covariant.

3.6 Rezk types

3.7 Directed univalence

For discrete types A and B, we seek to make an identification between morphisms $\hom_{\mathcal{U}_{\text{cov}}}(A, B)$ and functions $A \to B$. Just as we do so in informal HoTT, we can make such an identification by defining some canonical function:

Lemma 3.8. *For any two discrete types A and B, there is a function*

$$\text{homtofun} : \hom_{\mathcal{U}_{\text{cov}}}(A, B) \to (A \to B). \tag{4}$$

Proof. We first observe that the type family

$$\zeta :\equiv \lambda(B : \mathcal{U}_{\text{cov}}).\, x : \mathcal{U}_{\text{cov}} \to \mathcal{U}_{\text{cov}} \tag{5}$$

is covariant, so by Lemma 3.7 the type family

$$C :\equiv \lambda(B : \mathcal{U}_{\text{cov}}).\, A \to \zeta(B) : \mathcal{U}_{\text{cov}} \to \mathcal{U}$$

is also covariant. Note that (5) means that $C \equiv \lambda(B : \mathcal{U}_{\text{cov}}).\, A \to B$. Finally, we appeal to the Yoneda lemma to define homtofun:

$$\text{homtofun} :\equiv \text{yon}^C_A(\text{id}_A)(B) :\equiv \lambda f.\, f_*(\text{id}_A) : \hom_{\mathcal{U}_{\text{cov}}}(A, B) \to (A \to B)$$

□

Remark. Note that homtofun acts functorially, as for any $f : \hom_{\mathcal{U}_{\text{cov}}}(A, B)$ and $g : \hom_{\mathcal{U}_{\text{cov}}}(B, C)$,

$$\begin{aligned} &\text{homtofun}(g \cdot f) \equiv \\ &\quad (g \cdot f)_*(\text{id}_A) = \\ &\qquad g_*(f_*(\text{id}_A)) \end{aligned}$$

with the second equality following from Proposition 8.16.

Just as HoTT does not come equipped with tools to show idtoeqv is an equivalence, sHoTT also does not come equipped with the tools to show homtofun is an equivalence. Instead, we can postulate *directed univalence* as an axiom.

Axiom (Directed Univalence)**.** *For any discrete types $A, B : \mathcal{U}_{cov}$, the function defined in Lemma 3.8 is an equivalence.*

For discrete types A and B, we can break the equivalence up into:

- An introduction rule for $\hom_{\mathcal{U}_{\text{cov}}}(A, B)$ denoted dua for "directed univalence axiom":
$$\text{dua} : A \to B \to (\hom_{\mathcal{U}_{\text{cov}}}(A, B))$$
- An elimination rule
$$\text{homtofun} :\equiv \text{yon}_A^C(\text{id}_A) : \hom_{\mathcal{U}_{\text{cov}}}(A, B) \to (A \to B)$$
- The propositional computation rule
$$\text{homtofun}(\text{dua}(f)) = f$$
- The propositional uniqueness rule, which states for any $f : \hom_{\mathcal{U}_{\text{cov}}}(A, B)$,
$$f = \text{dua}(\text{homtofun}(f))$$

Directed univalence also allows us to identify identity morphisms with identity functions and composition of morphisms in $\mathcal{U}_{\text{cov}}$ with composition of functions, stated tersely as

$$\begin{gathered}\text{idhom}_A = \text{dua}(\text{id}_A)\\ \text{dua}g \cdot \text{dua}f = \text{dua}(g \circ f).\end{gathered}$$

The first equality from the fact that $\text{id}_A = (\text{idhom}_A)_*(\text{id}_A) \equiv \text{homtofun}(\text{idhom}_A)$ by proposition 8.16. To show the second, if we define $p :\equiv \text{dua}(f)$ and $q :\equiv \text{dua}(g)$, then

$$\text{dua}(g \circ f) = \text{dua}(\text{homtofun}(p) \circ \text{homtofun}(q)) = \text{dua}(\text{homtofun}(p \cdot q)) = p \cdot q$$

Finally, the directed univalence even extends to characterizing dependent functions as morphisms in $\mathcal{U}_{\text{cov}}$.

Lemma 3.9. *For any type family $P : A \to \mathcal{U}_{cov}$, for all $x : A$ and $Q : \tilde{P} \to \mathcal{U}_{cov}$, there is an equivalence*

$$\prod_{p:P(x)} Q(x, p) \simeq \sum_{f:\hom_{\mathcal{U}_{\text{cov}}}(P(x), \sum_{(p:P(x))} Q(x,p))} \text{dua}(\text{fst}) \circ f = \text{idhom}_{P(x)},$$

where fst : $\sum_{(p:P(x))} Q(x, p) \to P(x)$

Proof. We begin by noting the following equivalences

$$\begin{aligned}
\sum_{f:\mathrm{hom}_{\mathcal{U}_{\mathrm{cov}}}(P(x),\sum_{(p:P(x))} Q(x,p))} \mathrm{dua}(\mathrm{fst}) \circ f = \mathrm{idhom}_{P(x)} &\simeq \\
\sum_{f:P(x)\to\sum_{(p:P(x))} Q(x,p)} \mathrm{dua}(\mathrm{fst}) \circ \mathrm{dua}(f) = \mathrm{idhom}_{P(x)} &\simeq \\
\sum_{f:P(x)\to\sum_{(p:P(x))} Q(x,p)} \mathrm{dua}(\mathrm{fst} \circ f) = \mathrm{idhom}_{P(x)} &\simeq \\
\sum_{f:P(x)\to\sum_{(p:P(x))} Q(x,p)} \mathrm{fst} \circ f = \mathrm{id}_{P(x)} &
\end{aligned}$$

with the first equivalence following from exercise 2.10, the second from the functorality of dua and the third coming from theorem 4.6.3. Now that we are working with functions, we can reduce the types further:

$$\begin{aligned}
\sum_{f:P(x)\to\sum_{(p:P(x))} Q(x,p)} \mathrm{fst} \circ f = \mathrm{id}_{P(x)} &\simeq \\
\sum_{(f_1,f_2):\sum_{(g:P(x)\to P(x))} \prod_{(p:P(x))} Q(x,g(p))} f_1 = \mathrm{id}_{P(x)} &\simeq \\
\sum_{(f_1:P(x)\to P(x))} \ \sum_{(f_2:\prod_{(p:P(x))} Q(x,f_1(p)))} f_1 = \mathrm{id}_{P(x)} &\simeq \\
\prod_{p:P(x)} Q(x,p) &
\end{aligned}$$

with the first equivalence following from the type theoretic axiom of choice, the second following from the associativity of sigma types and the third following from singleton contractibility. Thus, we can get our desired equivalence by composing the two equivalences formed in this proof. ☐

3.8 Inner fibrations

In this section, we introduce another important class of fibrations: *inner fibrations*. They extend the notion of the Segal property to the dependent case. That is, inner fibrations have to do with completing *horns* in the total space.

Definition 3.7. For any type A, type family $C : A \to \mathcal{U}_{\mathrm{cov}}$, and

$$x, y, z : A$$

$$f : \mathrm{hom}_A(x,y) \qquad g : \mathrm{hom}_A(y,z) \qquad h : \mathrm{hom}_A(x,z)$$

$$t : \mathrm{hom}^2_A\left(\begin{tikzcd}[row sep=small] & y \arrow[dr, dash, "g"] & \\ x \arrow[ur, dash, "f"] \arrow[rr, dash, "h"'] & & z \end{tikzcd}\right)$$

$$u : C(x) \qquad v : C(y) \qquad w : C(z)$$

$$p : \mathrm{hom}_{C(f)}(u,v) \qquad q : \mathrm{hom}_{C(g)}(v,w)$$

we say C is **inner** if the type

$$\sum_{r:\hom_{C(h)}(u,w)} \hom^2_{C(t)}\left(\begin{tikzcd}[sep=small] & v \arrow[dr, dash, "q"] & \\ u \arrow[ur, dash, "p"] \arrow[rr, dash, "r"'] & & w\end{tikzcd}\right) \tag{6}$$

is contractible.

When the base of a type family is Segal, we state the inner condition much more tersely.

Lemma 3.10 (BW23)**.** *Given a Segal type A, a type family $C : A \to \mathcal{U}$ is inner if and only if $\widetilde{C}$ is Segal.*

Covariant families are a wealthy source of inner families. Not only are covariant families a specialization of inner families, but the mapping space between covariant families, even dependently, is an inner type family as well.

Lemma 3.11 (BW21)**.** *Every covariant type family is an inner type family.*

Lemma 3.12. *Given a Segal type A, for any $P : A \to \mathcal{U}_{cov}$ and $Q : \widetilde{P} \to \mathcal{U}_{cov}$ the type family*

$$\lambda(x : A). \prod_{p:P(x)} Q(x,p)$$

is inner.

Proof. Since A is Segal, by lemma 3.10 it suffices to show

$$\sum_{(a:A)} \prod_{(p:P(a))} Q(a,p)$$

is Segal. Note that

$$\Lambda^2_1 \to \sum_{(a:A)} \prod_{(p:P(a))} Q(a,p) \simeq \sum_{(\phi:\Lambda^2_1 \to A)} \prod_{(t:\Lambda^2_1)} \prod_{p:P(\phi(t))} Q(\phi(t),p)$$

by the type theoretic axiom of choice. The type of extensions of some (ϕ,ψ) of this type is

$$\sum_{\mu:\left\langle \Delta^2 \to A \middle|^{\Lambda^2_1}_{\phi} \right\rangle} \left\langle \textstyle\prod_{t:\Delta^2} \prod_{(p:P(\mu(t)))} Q(\mu(t),p) \middle|^{\Lambda^2_1}_{\psi} \right\rangle .$$

Note that the base is contractible since A is Segal, hence this type of extensions is equivalent to

$$\left\langle \textstyle\prod_{t:\Delta^2} \prod_{(p:P(\mathrm{comp}_{g,f}(t)))} Q(\mathrm{comp}_{g,f}(t),p) \middle|^{\Lambda^2_1}_{\psi} \right\rangle \tag{7}$$

where g, f make up the legs of ϕ. We seek to show that this type is contractible. By lemma , letting

$$\zeta :\equiv \mathrm{comp}_{g,f}$$
$$\epsilon :\equiv \hom_{\mathcal{U}_{\mathrm{cov}}}(P(\zeta(t)), \sum_{p:P(\zeta(t))} Q(\zeta(t), p))$$
$$\mathrm{fst} : \sum_{p:P(\zeta(t))} Q(\zeta(t), p) \to P(\zeta(t))$$

to shorten definitions, this type is equivalent to

$$\left\langle \prod_{t:\Delta^2} \sum_{(f:\epsilon)} \mathrm{dua}(\mathrm{fst}) \circ f = \mathrm{idhom}_{P(\zeta(t))} \Big|^{\Lambda^2_1}_{\psi'} \right\rangle$$

which, by lemma 3.4 is equivalent to the extension type

$$\left\langle \prod_{t:\Delta^2} \sum_{(f:\epsilon)} \hom^2_{\mathcal{U}_{\mathrm{cov}}}(\mathrm{dua}(\mathrm{fst}), f; \mathrm{idhom}_{P(x)})) \Big|^{\Lambda^2_1}_{\psi''} \right\rangle$$

Let $\alpha \equiv \lambda t.\, [(P(\zeta(t)), \sum_{(p:P(\zeta(t)))} Q(\zeta(t), p), P(\zeta(t)), \mathrm{dua}(\mathrm{fst}), \mathrm{idhom}_{P(\zeta(t))}]$. The type above is then equivalent to

$$\left\langle \prod_{t:\Delta^2} \left\langle \Delta^2 \to \mathcal{U}_{\mathrm{cov}} \Big|^{\Lambda^2_2}_{\alpha(t)} \right\rangle \Big|^{\Lambda^2_1}_{\psi'''} \right\rangle$$

by lemma 2. We then proceed by applying which allows us to swap the order of the nested extension types:

define the extension type theorems so I can refer to the label here

$$\left\langle \prod_{s:\Delta^2} \left\langle \prod_{t:\Delta^2} \mathcal{U}_{\mathrm{cov}} \Big|^{\Lambda^2_1}_{\lambda t.\, \psi''(t)(s)} \right\rangle \Big|^{\Lambda^2_2}_{\lambda t.\, \alpha(t)} \right\rangle \tag{8}$$

Since $\mathcal{U}_{\mathrm{cov}}$ is Segal, the fibers of 8 are contractible. Thus, the entire type in 8 is contractible by relative functional extensionality. So, 7 is contractible as well, since it is equivalent to 8. □

4 Directed Higher Inductive Types

Unit

As a warmup, we define the type Unit, which only has one constructor:

- $\star$: Unit

Abstractly, this should be the type generated by a single point. By definition, we can always form the morphism $\mathrm{id}_\star : \hom_{\mathrm{Unit}}(\star, \star)$. We expect, though, for there to be no more information than we put into the type. So, the identity should be the only morphism of the unit type.

Lemma 4.1. *For all* $f : \hom_{\mathrm{Unit}}(\star, \star)$,

$$f = id_\star$$

Proof. For all $x : \mathbb{2}$, we have that $f(x) = \star = \mathrm{id}_\star(x)$. By functional extensionality, $f = \mathrm{id}_\star$. $\square$

The unit type corresponds to the category with one object and one morphism (the identity).

Directed Interval

For our first non-trivial example of a directed higher inductive type, we start with the **directed interval**. Let I be the *Segal* type generated by constructors

- $0_I : I$
- $1_I : I$
- $\mathrm{seg} : \hom_I(0_I, 1_I)$.

The recursion principle for I states that for any Segal type B with

- a point b_0
- a point b_1
- a morphism $s : \hom_B(b_0, b_1)$,

there is a function $f : I \to B$ such that $f(0_I) \equiv b_0$, $f(1_I) \equiv b_1$, and $f(\mathrm{seg}) = s$.

The induction principle for I states that for any inner type family $C : I \to U$ along with

- a point $c_0 : C(0_I)$
- a point $c_1 : C(1_I)$
- a dependent morphism $s : \hom_{C(seg)}(c_0, c_1)$,

there is a function $f : \prod_{(x:I)} C(x)$ such that $f(0_I) \equiv c_0$, $f(1_I) \equiv c_1$ and $f(\mathrm{seg}) = s$.

We can show that the recursion principle is derivable from the induction principle:

Lemma 4.2. *If B is a Segal type with $b_0 : B$, $b_1 : B$ and $s : \hom_B(b_0, b_1)$, then there is a function $f : I \to B$ satisfying*

$$\begin{aligned} f(0_I) &= b_0, \\ f(1_I) &= b_1, \\ f(seg) &= s \end{aligned}$$

Moreover, we can also prove a uniqueness principle:

Lemma 4.3. *If B is a Segal type and f, g are two maps along with equalities*

$$\begin{aligned} p &: f(0_I) =_B g(0_I), \\ q &: f(1_I) =_B g(1_I), \\ r &: f(seg); q = p; g(seg). \end{aligned}$$

Then for all $x : I$ we have $f(x) = g(x)$.

Not only does encode-decode allow us to characterize the path space of a given type, but the technique also allows us to characterize the morphisms of a given type. Since $\mathcal{U}_{\text{cov}}$ is Segal, we use the recursion principle for I to define a type family code : $I \to \mathcal{U}_{\text{cov}}$ with the defining equations:

$$\begin{aligned} \text{code}(0_I) &:\equiv \text{Void} \\ \text{code}(1_I) &:\equiv \text{Unit} \\ \text{code}(\text{seg}) &:= \text{dua}\,! \end{aligned}$$

we note that the covariance of code allows us to prove basic facts about such a morphism in I.

Lemma 4.4. $\hom_I(1_I, 0_I) \simeq \text{Void}$

Proof. For any $f : \hom_I(1_I, 0_I)$, we note that $\text{code}_*(f)(\star) : \text{Void}$, hence there is a term $\phi : \hom_I(1_I, 0_I)$, which is enough to say that $\hom_I(1_I, 0_I) \simeq \text{Void}$ □

We seek to show that code is represented by the representable $\hom_A(1_I, -)$. With that in mind, we want to exhibit a fiberwise equivalence between the representable and code. So, we start by defining the fiberwise maps encode and decode.

Lemma 4.5. *There is a function*

$$encode : \prod_{(x:I)} \hom_I(1_I, x) \to \text{code}(x)$$

Proof. We can define encode by appealing to the Yoneda lemma:

$$\text{encode} :\equiv \text{yon}_{1_I}^{\text{code}}(\star)$$

□

Since code and $\lambda x.\hom_I(1_I, x)$ are covariant and I is Segal, we know that $\lambda x.\text{code}(x) \to \hom_I(1_I, x)$ is inner. With that, and lemma 0.3, we can define decode.

Lemma 4.6. *There is a function*

$$decode : \prod_{(x:I)} code(x) \to \hom_I(1_I, x).$$

Proof. First, we define functions for the point constructors:

$$\begin{aligned} f &:\equiv \lambda\, u.\mathrm{abort}(u) \\ g &:\equiv \lambda\_.\mathrm{id}_{1_I} \end{aligned}$$

If we want to define decode(0_I) as f and decode(1_I) as g, to define decode(seg), we need to produce a term of type

$$\hom_{(\lambda x.\mathrm{code}(x)\to \hom_I(1_I,x))(\mathrm{seg})}(f,g)$$

We can take advantage of directed univalence and ask instead for a term

$$\hom_{(\lambda x.\hom_{U_{\mathrm{cov}}}(\mathrm{code}(x),\hom_I(1_I,x)))(\mathrm{seg})}(\mathrm{dua}f,\mathrm{dua}g).$$

Since code and $\hom_I(1_I,\_)$ are covariant, we have functions $\mathrm{code}_*(\mathrm{seg})$ and seg_*.

$$\begin{aligned} &\mathrm{dua}(\mathrm{seg}_*)\cdot \mathrm{dua}(f) = && (9)\\ &\mathrm{dua}(\mathrm{seg}_* \circ f) = && (10)\\ &\mathrm{dua}(g\circ \mathrm{code}_*(\mathrm{seg})) = && (11)\\ &\mathrm{dua}(g)\cdot \mathrm{dua}(\mathrm{code}_*(\mathrm{seg})) && (12) \end{aligned}$$

With the second equality coming from the fact that $\mathrm{seg}_* \circ f = g\circ \mathrm{code}_*(\mathrm{seg})$ by functional extensionality. The equalities (3) and (5) are associated with 2-simplexes witnessing the equality. We can take the pushout of the two 2−simplices associated with the equalities, which diagrammatically gives a commutative square:

$$s' = \begin{tikzcd} \hom I(1_I,0_I) \arrow[r, "\mathrm{dua}(\mathrm{seg}_*)"] & \hom_I(1_I,1_I) \\ \mathrm{code}(0_I) \arrow[u, "\mathrm{dua}f"] \arrow[r, "\mathrm{dua}(\mathrm{code}_*(\mathrm{seg}))"'] & \mathrm{code}(1_I) \arrow[u, "\mathrm{dua}g"'] \end{tikzcd}$$

From which, we can define a morphism:

$$\lambda i.\mathrm{dcoe}(s(i)) : \hom_{\lambda x.\mathrm{code}(x)\to\hom_I(1_I,x)}(\mathrm{dcoe}(\mathrm{dua}f),\mathrm{dcoe}(\mathrm{dua}g))$$

which, since dua and dcoe are quasi-inverses, yields a morphism

$$s : \hom_{\lambda x.\mathrm{code}(x)\to\hom_I(1_I,x)}(f,g),$$

as desired. Thus, we define decode as follows:

$$\begin{aligned} \mathrm{decode}(0_I) &= f\\ \mathrm{decode}(1_I) &= g\\ \mathrm{decode}(\mathrm{seg}) &= s. \end{aligned}$$

□

With encode and decode defined, we can show that they form an equivalence.

Lemma 4.7. *There is a map*

$$\eta : \Pi_{x:I}\Pi_{f:\hom_I(1_I,x)}\mathrm{decode}\,x(\mathrm{encode}\,xf) = f$$

Proof. Noting that $\lambda x.\mathrm{decode}\,x(\mathrm{encode}\,xf) = f$ is covariant by theorem 8.26 in op. cit., we can further apply theorem 9.5 (dependent Yoneda lemma) in op. cit. So, it suffices to provide a term of the type

$$\mathrm{decode}\,1_I(\mathrm{encode}\,1_I\mathrm{id}_{1_I}) = \mathrm{id}_{1_I}.$$

Note, though, that encode was defined using the Yoneda lemma, hence it is a natural transformation that sends id_{1_I} to $\star$. So, our equality becomes

$$\mathrm{decode}\,1_I \star = \mathrm{id}_{1_I}.$$

We know how decode acts on these inputs, so it suffices to provide a term of the type

$$\mathrm{id}_{1_I} = \mathrm{id}_{1_I}$$

of which $\mathrm{refl}_{\mathrm{id}_{1_I}}$ suffices. □

Lemma 4.8. *The type family* $\lambda(x,u) : \tilde{\mathrm{code}}\,.\mathrm{encode}\,x\,(\mathrm{decode}\,x\,u) =_{\mathrm{code}(x)} u$ *is covariant.*

Proof. Let $(x,u),(y,v) : \tilde{\mathrm{code}}$ such that there is a morphism from (x,u) to (y,v). Since code is covvariant, it also suffices to consider a morphism $f : \hom_I(x,y)$ and $\mathrm{trans}_{f,u} : \hom_{\lambda x.\mathrm{code}(x)(f)}(u,v)$. We can define a morphism

$$f_t :\equiv \lambda\, s.\ \bigwedge_f(t,s) : \hom_I(x, f(t)).$$

Thus, given a point $\mu : \mathrm{encode}\,x\,(\mathrm{decode}\,x\,u) =_{\mathrm{code}(x)} u$, the morphism

$$\phi' :\equiv \lambda t : \Delta^1.(f_t)_*(\mu)$$

has source μ and target $f_*(\mu) : f_*(\mathrm{encode}\,x\,(\mathrm{decode}\,x\,u)) =_{\mathrm{code}(y)} f_*(u)$. Since $\lambda x\,\lambda u\,.\mathrm{encode}\,x\,(\mathrm{decode}\,x\,u)$ is a fiberwise map between covariant types, ϕ' induces a map ϕ which has source μ which has type

$$\mathrm{encode}\,x\,(\mathrm{decode}\,x\,u)) =_{\mathrm{code}(x)} u$$

and a target which has type

$$\mathrm{encode}\,y\,(\mathrm{decode}\,y\,f_*(u))) =_{\mathrm{code}(y)} f_*(u).$$

That is, ϕ is a map that lives over f and $\mathrm{trans}_{f,u}$. If there were another map ψ, then for every $t : \Delta^2$,

$$\phi(t) : \mathrm{encode}\,f(t)\,(\mathrm{decode}\,f(t)\,(f_t)_*(u))) =_{\mathrm{code}(f(t))} (f_t)_*(u)$$

and

$$\psi(t) : \mathrm{encode}\, f(t)\, (\mathrm{decode}\, f(t)\, (f_t)_*(u))) =_{\mathrm{code}(f(t))} (f_t)_*(u)$$

Note though, that

$$\mathrm{encode}\, f(t)\, (\mathrm{decode}\, f(t)\, (f_t)_*(u))) =_{\mathrm{code}(f(t))} (f_t)_*(u) \simeq$$
$$\mathrm{hom}_{\lambda x.\mathrm{code}(x)(f_t)}(\mathrm{encode}\, x\, (\mathrm{decode}\, x\, u), (f_t)_*(u))$$

Since the latter type is contractible, we are done. □

Lemma 4.9. *There is a map*

$$\epsilon : \Pi_{x:I}\Pi_{u:code(x)}\mathrm{encode}\, x(\mathrm{decode}\, xu) = u$$

Proof. First, we note that the type family $\lambda x.\Pi_{u:\mathrm{code}(x)}\mathrm{encode}\, x(\mathrm{decode}\, xu) = u$ is inner due to previous lemmas. Thus, we can define ϵ as follows:

$$\epsilon(0_I) = \lambda u.\mathrm{abort}(u)$$
$$\epsilon(1_I) = \lambda \star .\mathrm{refl}_\star$$

with ϵ(seg) proceeding like decode(seg). □