Encode-decode for directed higher inductive types

by

Chukwuebuka Akubilo
Class of 2025

A thesis submitted to the
faculty of Wesleyan University
in partial fulfillment of the requirements for the
Degree of Bachelor of Arts
with Departmental Honors in Computer Science

As an empiricist I continue to think of the conceptual scheme of science as a tool, ultimately, for predicting future experience in the light of past experience. Physical objects are conceptually imported into the situation as convenient intermediaries—not by definition in terms of experience, but simply as irreducible posits comparable, epistemologically, to the gods of Homer. Let me interject that for my part I do, qua lay physicist, believe in physical objects and not in Homer's gods; and I consider it a scientific error to believe otherwise. But in point of epistemological footing the physical objects and the gods differ only in degree and not in kind. Both sorts of entities enter our conception only as cultural posits. The myth of physical objects is epistemologically superior to most in that it has proved more efficacious than other myths as a device for working a manageable structure into the flux of experience.

Two Dogmas of Empiricism, W. V. O. Quine

Acknowledgements

I must first thank my advisor Dan Licata. Your guidance and understanding, both during your class and throughout my research under you, have been invaluable. Throughout my constant misunderstandings in the incredibly slow process that is learning, I never once felt judged but rather encouraged. Your clarity of thought, command of logic, and graceful attitude are deeply inspiring. I am especially grateful for your patience during many meetings, even when my thoughts were scattered or incomplete. Talking with you often revealed how much more I had to learn, and those moments were pivotal for my growth. Thank you for having trust in me and allowing me autonomy in choosing my topic, I truly cannot overstate how working with you has changed the direction of my life for the better.

I also extend my gratitude to Robert Rose. I appreciate all the knowledge sharing sessions we had, and you made me truly understand the level of work and effort required to understand things deeply. Watching you solve problems has allowed me to see that things do not always have to be difficult or scattered. Your diligence and astuteness left a lasting impression on me, and I aspire to embody these qualities in my own work one day.

Trey, I owe you thanks for being a great friend and a constant presence in the CS lab, where I spent countless hours. I'm incredibly grateful for our early work sessions, even when we were slightly lost on something as simple as a derivation tree. Reflecting on how far we've come brings me joy. Our conversations—ranging from AI alignment, random philosophy, and the news of the day to type theory and our respective projects—kept me grounded and motivated throughout this journey.

Finally, thank you, Selena, for being my rock. Your unwavering support, not just during this thesis but throughout college, has been a constant source of strength. You have been a true friend and an anchor for me. I genuinely could not have done this without you by my side. Thank you. A million thank yous.

Abstract

Homotopy Type Theory (HoTT) extends Martin-Löf Type Theory (MLTT) with new axioms and type constructors that facilitate the study of ∞-groupoids and enable a homotopical interpretation of type theory. In this interpretation, the equality type is viewed as a *path* in a topological space, and type families are considered fibrations. Central to this framework is *Voevodsky's univalence axiom*, which formalizes the mathematical shorthand that identifies isomorphic objects as equal. This axiom allows equivalences between types ($A \simeq B$) to be lifted along type families to equivalences ($C(A) \simeq C(B)$). Additionally, HoTT incorporates *higher inductive types*, which are defined inductively using both point and *path* constructors, enabling the construction of free groupoids such as the interval, the circle, homotopy pushouts, and colimits.

Simplicial Type Theory (STT) builds on HoTT by introducing additional layers for constructing polytopes out of products of the directed interval $\mathbb{2}$. This extension allows for the exploration of categorical structures, including morphisms, 2-morphisms, and higher morphisms. This thesis assumes a classifying universe for covariant fibrations ($\mathcal{U}_c$) and *directed univalence* to define and prove statements about *directed higher inductive types*. The universe $\mathcal{U}_c$ enables the treatment of type families with codomain $\mathcal{U}_c$ as covariant fibrations. Directed univalence generalizes the univalence axiom by formalizing the notion that morphisms between types are functions, thus facilitating the study of concrete categories within type theory. Directed higher inductive types extend the inductive type schema to include *morphism* constructors, supporting the construction of free categories.

This thesis explores the construction of free categories such as the directed interval and the simplicial circle. These extensions highlight the potential of STT to provide a robust framework for studying categorical structures in a homotopical and directed context.

Contents

Introduction

Martin-Löf Type Theory (MLTT), alternatively referred to as Intuitionistic Type Theory, is a constructivist formal logic system that presents a distinct foundation compared to set theory. A key difference lies in its primitive notion of a collection: instead of *sets*, MLTT utilizes *types*. These types, unlike sets, operate under different principles. Crucially, MLTT leverages the Curry-Howard isomorphism to establish a connection between mathematical logic and programming languages. This isomorphism manifests in the "proofs as programs" paradigm, where *type specifications* are understood as *propositions* of formal logic. Consequently, the typing judgement

$$a : A$$

is also read as

$$a \text{ is a proof of the proposition } A.$$

This connection between types and propositions enables *proof-relevant mathematics*, a framework for manipulating concrete "proof objects" not found in classical mathematics. The analogy doesn't stop there. A further essential element of type theory is the *type-indexed family of types*, often shortened to *family* or *type family*, whose typing judgement is expressed as:

$$x : A \vdash B(x) : \mathcal{U}.$$

Type families can be interpreted as representing a *hypothetical judgement*, asserting that $B(x)$ is a proposition under the assumption $x : A$.

Another fundamental distinction between MLTT and classical mathematics lies in the nature of equality. In MLTT, equality manifests in two distinct forms. The first pertains to computation. We write

$$a \equiv b : A$$

and say that a **is definitionally equal to** b. This form of equality is inherent and holds *a priori* by definition, requiring no evidence. Crucially, definitional equality allows for direct substitution: if $a \equiv b$, then we immediately know that $B(a) \equiv B(b)$.

The second form of equality addresses *provability*. In many situations, particularly in mathematics, we desire to consider objects with the same fundamental properties as being equal, even if they are not definitionally identical. A prime example is category theory, where equality extends beyond simple definition. It is standard practice to treat *equivalent* objects as equal, a concept reflected in the fact that most categorical definitions hold "up to canonical isomorphism," or *equivalence*. Consider, for instance, sets C and D within the category of sets (Set), and the Cartesian product operation $\times$. While technically distinct, the product sets $C \times D$ and $D \times C$ are "functionally" the same, both satisfying the universal property of the product. Similarly, one might wish to consider the sets $C \times \mathbb{1}$, $\mathbb{1} \times C$, and C as representing the same underlying object.

However, attempting to directly impose a *monoid* structure on the category of sets based on strict equality immediately encounters a problem: the sets C and $C \times \mathbb{1}$ are not, in fact, equal. To navigate these situations where strict equality is too restrictive, we develop more flexible notions of "sameness" and construct our definitions accordingly. As previously mentioned, in category theory, the concept of *equivalence* precisely captures this desired notion of sameness. Therefore, categories that satisfy the group axioms not by strict equality but "up to equivalence" are termed *symmetric monoidal categories*, and (Set) provides a fundamental example of such a structure.

As a built-in feature, type theory offers a solution for representing these more flexible notions of "sameness" through *identity types*. Specifically, for any two terms within a type, type theory provides a type whose elements are precisely the proofs of equality between those terms. We write the type of proofs witnessing the identity of two terms as

$$x =_A y : \mathcal{U},$$

interpret this as ""x **is propositionally equal to** y", and call an inhabitant $p : x =_A y$ a **path** from x to y. These identity types truly embody the notion of equality, as evidenced by the definability of terms demonstrating that the identity type satisfies the axioms of an equivalence relation:

$$\mathsf{refl}_{\_} : \prod_{x:A} a =_A a,$$

$$(\text{-})^{-1} : \prod_{x,y:A} (x =_A y) \to (y =_A x),$$

and

$$\text{-} \circ \text{-} : \prod_{x,y,z:A} (x =_A y) \to (y =_A z) \to (x =_A z).$$

These terms form data witnessing that identity types are reflexive, symmetric, and transitive respectively.

One of the built-in strengths of our type theory is Leibniz's Law, also known as the indiscernibility of identicals. This means that given a proof of equality $p : x =_A y$, we can naturally transform elements of the type family $B(x)$ into elements of $B(y)$. This transformation, known as a coercion, is not just a simple function; it is a full equivalence, $B(x) \simeq B(y)$.

Type theory also exhibits more explicit connections with category theory. Notably, Hofmann and Streicher's observation [HS98] revealed that MLTT types possess the structure of groupoids – categories where every morphism is an isomorphism. Under this interpretation, a $p : x =_A y$ (and any path propositionally equal to p) corresponds to a morphism $f : \mathsf{hom}_A(x, y)^*$. Essentially, morphisms are equivalence classes of paths, where the equivalence relation is propositional equality. Furthermore, identity types satisfy coherence rules like unit laws and associativity, ensuring that this interpretation of types

*This is meant categorically, not type theoretically.

as groupoids within a category is well-founded. However, this groupoid interpretation doesn't fully capture the richness of type theory. For instance, given three paths $p : x =_A y$, $q : y =_A z$, and $r : z =_A w$, there is an identification

$$(p \cdot q) \cdot r =_{\mathcal{U}} p \cdot (q \cdot r).$$

While these two morphisms are not definitionally equal, they are identified as the same morphism due to our quotienting operation under propositional equality. This discrepancy strongly suggests the presence of a richer, higher-dimensional structure within types. Indeed, this observation motivates the interpretation of types as weak ∞-groupoids [vdBG10, AW09, GG08, KL18, War08], a more sophisticated notion than simple groupoids. An ∞-groupoid can be informally understood as a category where morphisms possess a dimension, with higher-level morphisms constructed from lower ones, and crucially, all morphisms at every level are invertible. To illustrate this dimension property, we can borrow topological language: objects are 0-cells, morphisms between objects are 1-cells, morphisms between 1-cells are 2-cells, and so on, ad infinitum.

Beyond the inherent structure of types, two fundamental questions remain:

1. Under what conditions does an equivalence of types, $B(x) \simeq B(y)$, imply an equality of types, $B(x) =_{\mathcal{U}} B(y)$?

2. How can we formally define basic *free* ∞-groupoids arising in topology, such as the circle, torus, and cell complexes in general?

While MLTT lacks the expressive power to generally answer these questions, extending it with Voevodsky's univalence axiom and higher inductive types provides a positive resolution. This extended system, known as Homotopy Type Theory (HoTT), directly addresses these limitations. Voevodsky's univalence axiom formalizes the principle that "equivalent structures are equal." Consequently, we can assert that type families indeed act functorially with respect to propositional equality. Specifically, univalence allows us to conclude that an equivalence $B(x) \simeq B(y)$ implies an equality $B(x) = B(y)$. Furthermore, higher inductive types enhance the inductive type schema by allowing for *path* constructors, not just point constructors. This provides a logical framework for defining common topological spaces like the circle and torus mentioned earlier, as well as more complex constructions such as homotopy pushouts and colimits.

The introduction of univalence and higher inductive types to MLTT, forming HoTT, is specifically intended to guide the interpretation of types as reflecting the homotopy structure of topological spaces. This foundational choice naturally leads to a homotopical interpretation of the type theory. A central concept within homotopy theory is that of a *fibration*. In essence, a fibration $p : E \to B$ is a special kind of map between spaces characterized by certain "lifting" properties, crucially the ability to lift paths in the base space B to corresponding paths in the total space E. Therefore, in HoTT, we consider type families $P : A \to \mathcal{U}$ as the type-theoretic counterparts of *fibrations*. This perspective is grounded in the fact that type families, like fibrations, induce maps from their

total space $\sum_{(x:A)} P(x)$ down to the base space A, and characteristically exhibit the path lifting property.

All of this machinery culminates in the view that HoTT provides a language for doing homotopy theory synthetically, without building up point-set topology first. To illustrate this synthetic approach, consider the calculation of the fundamental group of the circle. In algebraic topology, this calculation is a significant undertaking, requiring the development of homotopy lifting properties and the theory of covering spaces, all within the framework of point-set topology. The traditional method hinges on demonstrating that the winding map $w : \mathbb{R} \to S^1$ is a fibration, a property which allows one to relate loops in the circle to their "windings" in the real line. The proof of this fibration property and the subsequent calculations often involve meticulous manipulation of open sets, limits, and other point-set constructions.

Alternatively, in type theory, the calculation is strikingly direct [LS13]. After defining the circle S^1 as a higher inductive type with a basepoint $\mathsf{base} : S^1$ and a path $\mathsf{loop} : \mathsf{base} = \mathsf{base}$, we can define a type family $\mathsf{code} : S^1 \to \mathcal{U}$ by specifying its values on the constructors of the circle:

$$\begin{aligned} \mathsf{code}(\mathsf{base}) &:\equiv \mathbb{Z} \\ \mathsf{code}(\mathsf{loop}) &:\equiv \mathsf{ua}(\mathsf{succ}) \end{aligned}$$

Here, $\mathbb{Z}$ represents the integers, succ is the successor function on the integers, and ua (from the univalence axiom) lifts this function to an equivalence between types. Crucially, in HoTT, any function between types can be viewed as a fibration, so we get the structure of a fibration "for free". From this, we construct maps $\mathsf{encode} : \prod_{(x:S^1)}(\mathsf{base} = x) \to \mathsf{code}(x)$ and $\mathsf{decode} : \prod_{(x:S^1)} \mathsf{code}(x) \to (\mathsf{base} = x)$ that witness the equivalence between the loop space of the basepoint and the fiber of the type family at the basepoint.

For encode, given a point $x : S^1$ and a path $p : \mathsf{base} = x$, we define $\mathsf{encode}(x, p) :\equiv \mathsf{transport}^{\mathsf{code}}(p, 0)$. Here, $\mathsf{transport}^{\mathsf{code}}(p, -)$ describes how elements of $\mathsf{code}(\mathsf{base})$ are transported along the path p to elements of $\mathsf{code}(x)$. Since $0 : \mathsf{code}(\mathsf{base})$, applying $\mathsf{transport}^{\mathsf{code}}(p, -)$ to 0 yields an element of $\mathsf{code}(x)$. Intuitively, encode translates a loop based at base into an integer representing its winding number.

Conversely, decode maps elements of $\mathsf{code}(x)$ back to paths based at base. While its definition involves pattern matching on the constructors of the circle, the core idea is to map integers to corresponding loops in the circle.

Showing that encode and decode are inverses requires the induction principle of S^1 and the properties of transport, as well as the type of equational reasoning about paths that is central to HoTT. Finally, the fundamental group of the circle is calculated by observing that the loop space at the basepoint is equivalent to the fiber of the type family at the basepoint, which is equivalent to the integers: $\mathsf{base} = \mathsf{base} \simeq \mathsf{code}(\mathsf{base}) \equiv \mathbb{Z}$. This demonstrates that the fundamental group of the circle is indeed the integers, a result derived directly and synthetically within the framework of HoTT.

HoTT, despite its strengths, faces challenges when dealing with asymmetric

relationships. To illustrate this concretely, we draw upon examples from [DR11, GWB24] and consider the formalization of an inductive family we'll call List. As a starting point, we define a monoid as a type A together with a binary operation $\star : A \times A \to A$ satisfying associativity and having an identity element. This can be formally expressed as:

$$\mathsf{monoid} :\equiv \sum_{(A:\mathcal{U})} \sum_{(\star:A\times A\to A)} \mathsf{associativity}(A,\star) \times \mathsf{identity}(A,\star)$$

with $\mathsf{associativity}$ defined as

$$\mathsf{associativity} :\equiv \prod_{(A:\mathcal{U})} \prod_{(\star:A\times A\to A)} \prod_{(a,b,c:A)} a \star (b \star c) = (a \star b) \star c$$

and $\mathsf{identity}$ defined as

$$\mathsf{identity} :\equiv \prod_{(A:\mathcal{U})} \prod_{(\star:A\times A\to A)} \sum_{(e:A)} \prod_{(a:A)} (e \star a = a) \times (a \star e = a).$$

As the names imply, $\mathsf{associativity}$ and $\mathsf{identity}$ express that its input satisfies associativity and has an identity, respectively. Having established the definition of a monoid, we now introduce, for each monoid $A : \mathsf{monoid}$, an inductive type $\mathsf{List}\ A$ generated by:

- $\mathsf{nil} : \mathsf{List}\ A$
- $\mathsf{cons}(a, l) : \mathsf{List}\ A$, for every element $l : \mathsf{List}\ A$ and $a : \mathsf{fst}(A)$.

We then define a function $\mathsf{sum} : \prod_{(A:\mathsf{monoid})} \mathsf{List}\ A \to A$ using List recursion. With this setup, we can ask: what structures does sum preserve? Demonstrating that sum preserves equality is straightforward; for instance, if $A = B$, then $\mathsf{sum}\ A = \mathsf{sum}\ B$. However, our mathematical intuition leads us to conjecture a broader claim: that sum also preserves preorders, which are characterized by reflexivity and transitivity.

To attempt to prove our conjecture within Homotopy Type Theory (HoTT), we might initially consider the types monoid and $\mathsf{List}\ A$ as categories, as defined in [Pro13, 9]. However, categories in HoTT are not a primitive notion but rather an analytic construction. Therefore, to even work with the category monoid, the type of morphisms $\mathsf{hom}_{\mathsf{monoid}}(A, B)$ must be explicitly defined and shown to satisfy the category axioms. Assuming we can establish this, and given a monoid morphism $f : \mathsf{hom}_{\mathsf{monoid}}(A, B)$, we then encounter the issue that the type of morphisms between the lifted structures, $\mathsf{hom}\ (\mathsf{List}\ A, \mathsf{List}\ B)$, is not automatically inhabited. Consequently, proving that List acts as a functor, as defined in *op. cit.*, becomes a necessary step. Moreover, while we might hope to leverage the parametricity of sum to obtain a proof of our conjecture as a "theorem for free" [Wad89], articulating this theorem requires first defining a map $f_* : \mathsf{List}\ A \to \mathsf{List}\ B$.

Once such a map f_* is defined, the corresponding "free theorem" would ideally be an element of the type $\mathsf{sum}B \circ f_* = f \circ \mathsf{sum}A$. However, despite

the theoretical possibility of these constructions, the inherent focus on symmetric relations within our type theory prevents a natural and direct handling of asymmetric relations like preorders.

To address this limitation, *directed type theory* has been investigated. These type theories generally fall into two primary variations. The first approach operates from the perspective that *every* type corresponds to an $(\infty, 1)$-*category*[†], rather than an ∞-groupoid. Examples of such type theories include:

1. The 2-dimensional type theory (2DTT) of Licata and Harper [DR11] addresses asymmetric relations by requiring explicit *variance* annotations on types and terms. This system enforces that covariantly introduced terms are used only covariantly, and similarly for contravariant terms, enabling statements like "Π is covariant in its range and contravariant in its domain." 2DTT diverges from MLTT by omitting the identity type constructor, instead employing a *term transformation* judgment for asymmetric relations. This is coupled with a coercion operation along type families, acting as a unidirectional analog of transport in HoTT. Defining a type in 2DTT involves specifying its formation rule, term introduction/elimination rules ($M : A$), transformation introduction/elimination rules ($\alpha : M \implies {}_A M'$), and coherence data with the type theory's substitution mechanism.

2. Building upon 2DTT, the directed type theory of Andreas Nuyts [Nuy15] extends the base type theory to incorporate the full power of HoTT, rather than just MLTT without identity types. A key feature of Nuyts' theory is the explicit articulation of four possible relationships between $f(a)$ and $f(b)$ for a morphism $a \to_A b$ and a function $f : A \to C$: invariance (no relation), isovariance ($f(a) =_C f(b)$), covariance ($f(a) \to_C f(b)$), and contravariance ($f(b) \to_C f(a)$). This results in four variances to track, compared to the two in 2DTT.

3. In contrast to 2DTT's reliance on a morphism judgment, Page North's type theory [Nor19] introduces a dedicated morphism type. This shift allows morphisms to be treated as first-class citizens within the type theory, enabling the formation of types of morphisms. Furthermore, it provides elimination principles for this morphism type, offering a directed way to reason about morphisms, akin to a directed form of identity elimination. However, similar to 2DTT, the burden of managing variance remains with the user.

In contrast to the view that all types correspond to $(\infty, 1)$-categories, the second perspective posits that only *some* types inherently possess the structure of ∞-categories. Consequently, in such type theories, the need for explicit variance annotations and manual variance management is eliminated. Currently, these

[†]An $(\infty, 1)$-category is a category with an infinite hierarchy of morphisms, where each layer is enriched by the last. Crucially, beyond dimension one, all n-morphisms (for $n > 1$) are invertible. For brevity, we will refer to $(\infty, 1)$-categories as ∞-categories.

approaches are primarily derived from Simplicial Type Theory (STT) [RS17], which will be the central focus of this thesis.

STT advances the landscape of directed type theory through the addition of a metatheoretic layer comprising of *shapes* and *extension types*. Shapes are conceptually thought of as the polytopes that are formed within cubes. Extension types can be thought of as a function type where the domain is a shape and which the behavior of maps on a particular subshape is known. This mechanism allows us to "carve out" specific structures within types, such as morphisms (1-cells) and 2-morphisms (2-cells), reflecting the simplicial nature of the theory. Furthermore, STT enables the formal specification of important type classes. For instance, Segal types, which encode a weak composition principle, can be defined, and their inhabitants naturally exhibit the coherence data characteristic of *categories*. Another crucial example is that of *covariant type families*. These families, given a morphism $f : \mathsf{hom}_A(a, b)$ and an element $u : C(a)$, provide a mechanism to lift the morphism f to a corresponding morphism $\mathsf{trans}_{f,u} : \mathsf{hom}_{C(b)}(u, f_*(u))$, effectively behaving as covariant fibrations.

As it stands, STT provides a strong foundation for addressing the formal category-theoretical questions that arise within higher category theory. This thesis leverages Simplicial Type Theory, assuming the existence of a directed univalence principle, to explore its consequences for the formalization of basic category theory. The primary contribution of this work is the examination of several directed higher inductive types, focusing on their construction and properties. We explore the construction of fundamental examples such as the directed interval and the simplicial circle, aiming to provide insights into their behavior within STT.

This thesis investigates the power and versatility of directed higher inductive types (DHITs) within the framework of simplicial type theory. In Section 1, we warm up with a recapitulation of Homotopy Type Theory (HoTT), providing context for the reader unfamiliar with the subject; however, we will not present any HoTT proofs already established in the literature, instead pointing the reader to [Pro13] for a thorough treatment.

Section 2 introduces the core machinery of simplicial type theory, largely following [RS17]. We begin with the foundational concepts of shapes and extension types and proceed to define fundamental type classes, including Segal, Discrete, and Rezk types, which correspond to pre-categories, groupoids, and univalent categories, respectively. We also introduce covariant type families, and, departing slightly from [RS17], we borrow from [BW22] to define inner type families, which lift 2-morphisms. Crucially, we introduce $\mathcal{U}_{\mathsf{cov}}$, our Segal directed univalent universe of discrete types, and prove several important lemmas that will be used in the subsequent section.

Finally, Section 3 constitutes the heart of this thesis, where we discuss DHITs. We demonstrate that DHITs, with their point, morphism, and Segal constructors allow us to specify free categories. Furthermore, we highlight the critical role of inner type families in preserving categorical structure when eliminating out of DHITs. We apply this framework to a detailed study of the directed interval and the simplicial circle, showcasing the practical utility of

DHITs. While we focus on these specific examples, the techniques developed herein, particularly the interplay between inner fibrations and DHIT elimination, are general and can be applied to a wider range of free categories, opening exciting avenues for future research in directed type theory.

1. HoTT

In this section, we delve into the foundational elements of HoTT, laying the groundwork for our exploration of directed type theory in subsequent sections. We will begin by examining the core concept of paths within types, which are fundamental to the homotopical interpretation of type theory. Building upon this, we will introduce homotopies, which provide a way to relate paths, and equivalences, a key notion for establishing sameness between types. These concepts are essential for understanding how HoTT captures higher-dimensional structures. We then explore how specific classes of types, namely sets and types representing propositions, are characterized within this framework. Finally, we will investigate properties of dependent pair types, also known as Σ-types, and establish some fundamental lemmas about them that will be useful in later sections.

1.1 Types are higher groupoids

The connection between type theory and topology is evident in the concept of paths. Paths in type theory, analogous to topological paths, act as morphisms in higher groupoids. The constant path at x is refl_x. Using path induction, we can show that path equality forms an equivalence relation, mirroring the behavior of topological paths. The following lemmas demonstrate this property.

Lemma 1.1.1 ([Pro13], 2.1.1)**.** *For any $x, y : A$ there is a function*

$$x = y \to y = x$$

*denoted by $p \mapsto p^{-1}$, such that $\mathsf{refl}_x^{-1} \equiv \mathsf{refl}_x$. We call p^{-1} the **inverse** of p.*

We can also show that path types are transitive.

Lemma 1.1.2 ([Pro13], 2.1.2)**.** *For any $x, y, z : A$ there is a function*

$$x = y \to y = z \to x = z$$

*denoted by $p \mapsto q \mapsto q \cdot p$. We call $q \cdot p$ the **concatenation** of p and q.*

However, in the proof-relevant setting of type theory, simply having functions demonstrating these equivalence relation properties is not enough to fully capture the higher groupoid structure. We need to ensure that these operations behave consistently with each other. This is where the concept of coherence laws comes in. These laws assert the equality of different ways of manipulating paths, reflecting the fact that proofs themselves can be considered equal.

Lemma 1.1.3 ([Pro13], 2.1.4)**.** *For any type A, $x, y, z, w : A$ $p : x = y$, $q : y = z$, and $r : z = w$ we have the following:*

1. *$p = p \cdot \mathsf{refl}_y$ and $p = \mathsf{refl}_x \cdot p$.*
2. *$p^{-1} \cdot p = \mathsf{refl}_y$ and $p \cdot p^{-1} = \mathsf{refl}_x$.*

3. $(p^{-1})^{-1} = p$.

4. $p \cdot (q \cdot r) = (p \cdot q) \cdot r$.

The lemmas presented so far illustrate the one-dimensional structure of paths. To fully realize the vision of types as higher groupoids, we must extend these concepts and demonstrate analogous properties for higher-dimensional paths – paths between paths, paths between paths between paths, and so on.

1.2 Functions are functors

A crucial aspect of Homotopy Type Theory is the principle that functions between types act consistently with the path structure, mirroring the behavior of functors in category theory. This means that functions not only map terms to terms but also transform paths between terms in a coherent way. To formalize this, we first establish that functions "respect equality," meaning they map equal terms to equal terms.

Lemma 1.2.1 ([Pro13], 2.2.1)**.** *For any function $f : A \to B$ and $x, y : A$, there is a function*

$$\mathsf{ap}_f : x = y \to f(x) = f(y)$$

such that each $x : A$, we have $\mathsf{ap}_f(\mathsf{refl}_x) \equiv \mathsf{refl}_{f(x)}$.

Building on this fundamental property, we can then demonstrate the full functorial behavior of functions on paths.

Lemma 1.2.2 ([Pro13], 2.2.2)**.** *For functions $f : A \to B$ and $g : B \to C$ and paths $p : x = y$ and $q : y = q$, we have:*

1. $\mathsf{ap}_f(p \cdot q) = \mathsf{ap}_f(p) \cdot \mathsf{ap}_f(q)$.

2. $\mathsf{ap}_f(p^{-1}) = (\mathsf{ap}_f(p))^{-1}$.

3. $\mathsf{ap}_g(\mathsf{ap}_f(p)) = \mathsf{ap}_{g \circ f}(p)$.

4. $\mathsf{ap}_{\mathsf{id}_A}(p) = p$.

1.3 Type families are fibrations

In the groupoid interpretation of type theory, a key goal is to establish the analogy between type families and fibrations from homotopy theory. We aim to treat a type family $P : A \to \mathcal{U}$ as a "bundle" of types over A, which requires demonstrating a suitable path lifting property. This property is facilitated by the concept of "transport", the type-theoretic equivalent of path lifting.

Lemma 1.3.1 ([Pro13], Transport, 2.3.1)**.** *For any type family $P : A \to \mathcal{U}$ and path $p : x =_A y$, there is a function*

$$\mathsf{transport}^P(p, -) : P(x) \to P(y),$$

or similarly denoted as $p_* : P(x) \to P(y)$ when clear from context.

Viewing a type family $P : A \to \mathcal{U}$ as the fibration $\sum_{(x:A)} P(x) \to A$, transport provides the mechanism for lifting paths. Given $u : P(x)$ and a path $p : x =_A y$, transporting u along p yields a point $p_*(u) : P(y)$. The path lifting property formalizes this, stating the existence of a path in the total space connecting (x, u) to $(y, p_*(u))$ that projects to the original path p.

Lemma 1.3.2 ([Pro13], Path lifting property, 2.3.2)**.** *Let $P : A \to \mathcal{U}$ be a type family and let $u : P(x)$ for some $x : A$. Then, for any $p : x = y$, there is a path*

$$\mathsf{lift}(u, p) : (x, u) = (y, p_*(u)).$$

in $\sum_{(x:A)} P(x)$ such that $\mathsf{fst}(\mathsf{lift}(u, p)) = p$.

1.4 Homotopies and equivalences

Beyond paths between terms, functions themselves can be related in type theory. A natural way to compare functions is through pointwise identification.

Definition 1.4.1 ([Pro13], 2.4.1)**.** Let $P : A \to \mathcal{U}$ be a type family and $f, g : \prod_{(x:A)} P(x)$. We write $f \sim g :\equiv \prod_{(x:A)} f(x) = g(x)$ and say an inhabitant of $f \sim g$ is a **homotopy** from f to g.

Since homotopies are essentially pointwise paths, we expect them to share fundamental properties with path equality.

Lemma 1.4.2 ([Pro13], 2.4.2)**.** *Homotopy is an equivalence relation on the dependent function type. That is, for any type family $P : A \to \mathcal{U}$, there are maps*

$$\prod_{f:\prod_{(x:A)} P(x)} f \sim f$$

$$\prod_{f,g:\prod_{(x:A)} P(x)} (f \sim g) \to (g \sim f)$$

$$\prod_{f,g,h:\prod_{(x:A)} P(x)} (f \sim g) \to (g \sim h) \to (f \sim h)$$

Moving beyond functions, we can consider how to express that two types, A and B, are "the same". In Homotopy Type Theory, simple path equality between types is insufficient to capture this notion fully. Drawing inspiration from category theory, the concept of **equivalence** provides a more robust notion of sameness.

A first attempt at defining equivalence might involve **quasi-inverses**. For a map $f : A \to B$, a quasi-inverse $g : B \to A$ would satisfy $f \circ g \sim \mathsf{id}_B$ and $g \circ f \sim \mathsf{id}_A$. The type of all quasi-inverses of f can be written as:

$$\mathsf{qinv}(f) :\equiv \sum_{g:B\to A} ((f \circ g \sim \mathsf{id}_B) \times (g \circ f \sim \mathsf{id}_A)).$$

While this notion aligns with the idea of equivalence in standard category theory, HoTT typically employs a different, more refined definition.

Instead of relying on a single quasi-inverse, we can decompose the notion of an inverse into two parts: left and right inverses. A function g is a **left inverse** of $f : A \to B$ if $g \circ f \sim \mathsf{id}_A$, and the type of all left inverses is:

$$\mathsf{linv}(f) :\equiv \sum_{g:B\to A} g \circ f \sim \mathsf{id}_A.$$

Similarly, g is a **right inverse** of f if $f \circ g \sim \mathsf{id}_B$, and the type of all right inverses is:

$$\mathsf{rinv}(f) :\equiv \sum_{g:B\to A} f \circ g \sim \mathsf{id}_B.$$

A map f is then considered an **equivalence** (or **bi-invertible**) if it possesses both a left and a right inverse. The type of proofs that f is an equivalence is defined as:

$$\mathsf{isequiv}(f) :\equiv \mathsf{biinv}(f) :\equiv \mathsf{linv}(f) \times \mathsf{rinv}(f).$$

This notion of equivalence generalizes the familiar idea of an isomorphism or bijection between sets. If A and B are sets, an equivalence $f : A \to B$ corresponds precisely to a function that is both surjective and injective. We can recover these concepts within type theory.

Definition 1.4.3 ([Pro13], 4.6.1)**.** For any $f : A \to B$

1. If for every $b : B$ the type $\left\|\mathsf{fib}_f(b)\right\|$ is inhabited, f is said to be **surjective**.
2. If for every $x, y : A$ the function $\mathsf{ap}_f : (x =_A y) \to (f(x) =_B f(y))$ is an equivalence, f is said to be an **embedding**.

The connection between equivalences, surjectivity, and embeddings is fundamental:

Lemma 1.4.4 ([Pro13], 4.6.3)**.** *A function $f : A \to B$ is an equivalence if and only if it is both surjective and an embedding.*

When working with type families, $P, Q : A \to \mathcal{U}$, we introduce **fiberwise maps** to relate them. A fiberwise map $f : \prod_{(x:A)} P(x) \to Q(x)$ provides a way to map elements within each fiber. Such maps naturally induce maps between the total spaces of the type families.

Definition 1.4.5 ([Pro13], 4.7.5)**.** For any type families $P, Q : A \to \mathcal{U}$ and fiberwise map $f : \prod_{(x:A)} P(x) \to Q(x)$, we define

$$\mathsf{total}(f) :\equiv \lambda(x, u).\,(x, f(x, u)) : \sum_{x:A} P(x) \to \sum_{x:A} Q(x) \tag{1}$$

We can further define a **fiberwise equivalence**. A fiberwise map $f : \prod_{(x:A)} P(x) \to Q(x)$ is a fiberwise equivalence if, for each $x : A$, the map $f(x) : P(x) \to Q(x)$ is an equivalence. This property is closely tied to whether the induced map on total spaces is an equivalence.

Lemma 1.4.6 ([Pro13], 4.7.7)**.** *For any type families $P, Q : A \to \mathcal{U}$ and fiberwise map $f : \prod_{(x:A)} P(x) \to Q(x)$, f is a fiberwise equivalence if and only if* $\mathsf{total}(f)$ *is an equivalence.*

Equivalences play a crucial role in transporting structure. If $e : A \to B$ is an equivalence, it allows us to relate total spaces over A and B.

Lemma 1.4.7 ([Pro13], Exercise 2.17)**.** *For any type family $C : A \to \mathcal{U}$, types A and B, and equivalence $e : A \to B$ with quasi-inverse e^{-1},*

$$\sum_{a:A} C(a) \simeq \sum_{b:B} C(e^{-1}(b)) \tag{2}$$

Finally, even path composition itself exhibits the structure of an equivalence.

Lemma 1.4.8 ([Pro13], Exercise 2.6)**.** *For any $p : x = y$, the function*

$$(p \cdot -) : (y = z) \to (x = z) \tag{3}$$

is an equivalence.

1.5 Sets and Logic

In the realm of type theory, the familiar notion of a set can be understood through a lens that emphasizes the identity of its elements. Specifically, a set in type theory is a type that is distinguished by its objects, possessing no additional "higher homotopical data". This leads to the following definition:

Definition 1.5.1 ([Pro13], 3.1.1)**.** A type A is said to be a **set** if the type

$$\mathsf{isSet(A)} :\equiv \prod_{(x,y:A)} \prod_{(p,q:x=_A y)} p =_{x=_A y} q$$

Building upon this foundation, type theory organizes types into a hierarchy based on their level of truncation, which indicates the highest dimension at which non-trivial homotopical information exists. Sets occupy the level of 0-types. Of particular interest are the types that reside "below" sets in this hierarchy: contractible types (-2 types), which contain precisely one object, and propositions (-1 types), which contain at most one object. We now define these fundamental types, which are essential for understanding the broader concept of n-types.

Definition 1.5.2 ([Pro13], 3.11.1)**.** A type A is said to be **contractible** if the type

$$\mathsf{isContractible(A)} :\equiv \sum_{(x:A)} \prod_{(y:A)} x =_A y$$

Definition 1.5.3 ([Pro13], 3.3.1)**.** A type A is said to be a **proposition** if the type

$$\mathsf{isProp(A)} :\equiv \prod_{x,y:A} x =_A y$$

HoTT also allows us tools to approximate types up to the propositional level, allowing for traditional concepts like the set theoretic subset to have analogues in type theory.

Definition 1.5.4. [Pro13], 3.7 For any type A, there is another type $\|A\|$ called the **propositional truncation of** A, defined by the constructors

- For all $a : A$ we have $|a| : \|A\|$.
- For all $x, y : \|A\|$ we have $x =_{\|A\|} y$

Apart from approximating types, we can extend our the definition of propositions and contractible types to higher dimensions. In a proposition, its identity types are contractible. We expect a set to be a type that is distinguished by its objects, thus it must have no non-trivial paths. Thus, a set is a type whose identity types are propositions. This pattern generalizes to all higher dimensions as seen in the following definition:

Definition 1.5.5 ([Pro13], 7.1.1)**.** For any type A, we say it is a -2 type if it is contractible and an $n+1$ type if its identity types are n types.

We also have a fundamental result that relates the truncation level of a type, the highest level of which there is non-trivial homotopical data, to its identity types.

Lemma 1.5.6 ([Rij22], 12.4.4)**.** *For any type A, if it is an n-type, then its identity types are n-types.*

A fundamental result in homotopy type theory is the contractibility of based path spaces. This result, often referred to as **singleton contractibility**, is formally stated as follows:

Lemma 1.5.7 ([Pro13], 3.11.8)**.** *For any type A and $a : A$, the type $\sum_{(x:A)} a = x$ is contractible.*

When working with Sigma types (dependent sum types), the property of contractibility in either the base type or the fiber types leads to significant simplifications in the type structure, up to type equivalence. This is a crucial tool for simplifying type expressions.

Lemma 1.5.8 ([Pro13], 3.11.9)**.** *For any type family $P : A \to \mathcal{U}$:*

1. *If each $P(x)$ is contractible, then $\sum_{(x:A)} P(x) \simeq A$.*
2. *If A is contractible with center of contraction $a : A$, then $\sum_{(x:A)} P(x) \simeq P(a)$.*

Finally, it is important to note that logical principles arise naturally within type theory, and are not solely confined to the definition of n-types. We present a principle that is logically weaker than the full axiom of choice from classical set theory, known as **the type theoretic axiom of choice**.

Lemma 1.5.9 ([Pro13], 1.7)**.** *For a type X and type families $A : X \to \mathcal{U}$ and $P : \prod_{(x:X)} A(x) \to \mathcal{U}$,*

$$\left(\prod_{(x:X)} \sum_{(a:A(x))} P(x,a)\right) \simeq \left(\sum_{(g:\prod_{(x:X)} A(x))} \prod_{(x:X)} P(x,g(x))\right) \tag{4}$$

1.6 Univalence

As mentioned in the introduction, HoTT builds on top of MLTT by identifying paths between types $A =_{\mathcal{U}} B$ with equivalences $A \simeq B$. We make the identification by first defining a map in one direction:

Lemma 1.6.1 ([Pro13], 2.10.1)**.** *For any types A, B, there is a function*

$$\mathsf{idtoequiv} : A =_{\mathcal{U}} B \to A \simeq B$$

defined by path induction.

From there, univalence appears as an axiom:

Axiom 1.6.2 ([Pro13], 2.10.3)**.** *For all types $A, B : \mathcal{U}$, the function* idtoequiv *is an equivalence.*

1.7 Π-types

Given a type A and a type family $B : A \to \mathcal{U}$, the dependent function type $\prod_{(x:A)} B(x)$ is the type of all dependent functions. That is, a function that takes some $x : A$ and returns a section $b(x) : B(x)$. Dependent functions are well known in MLTT and their behavior is largely the same in HoTT. The difference, though, lies with the following lemma.

Lemma 1.7.1 ([Pro13], 2.9.3)**.** *For all types A, type families $B : A \to \mathcal{U}$, and dependent functions $f, g : \prod_{(x:A)} B(x)$ the function*

$$\mathsf{happly} : f = g \to \prod_{x:A} f(x) = g(x)$$

defined by path induction is an equivalence.

In MLTT, theorem 1.7.1 cannot be proven. In HoTT, the lemma is a consequence of univalence.

1.8 Σ-types

For a type A and a type family $C : A \to \mathcal{U}$, the sigma type $\sum_{(x:A)} C(x)$ consists of dependent pairs (a, u) where $a : A$ and $u : C(a)$. When the type family C is constant, i.e., $C(x) = B$ for all $x : A$, the sigma type is equivalent to the Cartesian product, denoted $A \times B$. Throughout this thesis, we will utilize several fundamental properties of the dependent pair type. We begin by demonstrating the symmetry of the Cartesian product, revisiting our discussion from the introduction.

Lemma 1.8.1. *For any two types A and B, we have $A \times B \simeq B \times A$.*

Proof. We define a function $f : A \times B \to B \times A$ by $f((a, b)) :\equiv (b, a)$, and a function $g : B \times A \to A \times B$ by $g((b, a)) :\equiv (a, b)$. We can then show that f and g are inverses of each other. For any $(a, b) : A \times B$, we have $g(f(a, b)) \equiv g((b, a)) \equiv (a, b)$. Similarly, for any $(b, a) : B \times A$, we have $f(g(b, a)) \equiv f((a, b)) \equiv (b, a)$. Therefore, f and g form an equivalence between $A \times B$ and $B \times A$. □

Furthermore, the dependent pair type exhibits associativity, a property that holds generally for dependent sums.

Lemma 1.8.2 ([Pro13], Exercise 2.10)**.** *For any type $A : \mathcal{U}$ and type families $B : A \to \mathcal{U}$ and $C : (\sum_{(x:A)} B(x)) \to \mathcal{U}$, the following holds*

$$\left(\sum_{(x:A)} \sum_{(y:B(x))} C((x, y))\right) \simeq \left(\sum_{p:\sum_{(x:A)} B(x)} C(p)\right) \tag{5}$$

2. sHoTT

Simplicial homotopy type theory extends HoTT with *extension types* and two extra layers, turning it into a *type theory with shapes*. It is specialized to the case of a cube $\mathbb{2}$ and an inequality tope, including rules making the tope layer the coherent logic[‡] of a strict interval. The top geometric layers interact with the bottom type theory layer through extension types. Extension types allow us to introduce dependent functions whose domain is a shape. To introduce an extension type, a shape that includes into the domain shape, a term that depends on that shape must be provided. Then, the extension type can be thought of as extending the provided section from a partial domain to a full domain.

With extension types in tow, we can probe simplicial shapes in types. That is, we can probe morphisms, 2-morphisms, 3-morphisms and higher by defining the right extension type. Using familiar notions from HoTT, we can then define a class of types whose morphisms have a weak composition structure, *Segal types*. Segal types are a useful model of the notion of category. We can also define a type of classes that has no interesting simplicial data. These types are ones where the path types and morphism types are equivalent, they are dubbed *discrete types*. Extending this further, types whose *isomorphisms* are equivalent to its paths are dubbed *Rezk types*.

We can also reflect our situation in HoTT in sHoTT. In HoTT, all type families are fibrations. In sHoTT, *some* type families are fibrations. We can define the type of type families that lift morphisms covariantly and contravariantly, which are aptly dubbed *covariant* and *contravariant type families*. We are not limited to lifting morphisms, we call families that lift 2-morphisms *inner type families*.

As mentioned before, we are working in a bespoke simplicial homotopy type theory. Included in our extended sHoTT is a Segal classifying universe of covariant fibrations. We also include a directed univalence axiom which allows us to turn morphisms between discrete types into functions between them.

In this section, we give an informal introduction to sHoTT and develop machinery for use in Section 4.

2.1 Three-layer type theory with shapes

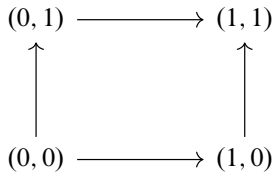
The foundation of sHoTT is a *three-layer type theory with shapes*. More specifically, it is a type theory that consists of a **cube layer** at the top, which is a simple type theory with finite products. Cubes in sHoTT are finite powers of $\mathbb{2}$, such as those in figure fig. 1. The cube layer is formally the rules

$$\mathbb{2}\ \mathsf{cube} \qquad\qquad 0 : \mathbb{2} \qquad\qquad 1 : \mathbb{2}$$

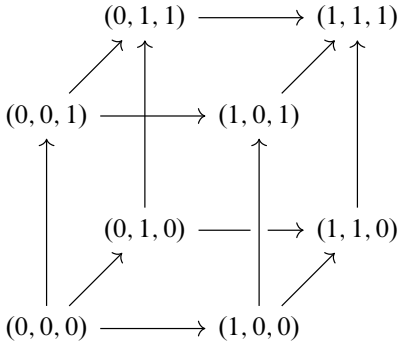
We have the **tope layer** sitting under the cube layer. The tope layer includes the binary relation $\leq$ along with axioms detailed in fig. 6 that turns $\mathbb{2}$ into the

[‡]A coherent logic is a fragment of first-order logic that only allows the quantifiers and connectives $\vee$, $\wedge$, $\top$, $\bot$, and $\exists$.

(a) $\mathbb{2}$

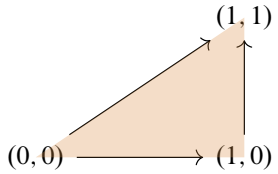


(b) $\mathbb{2} \times \mathbb{2}$

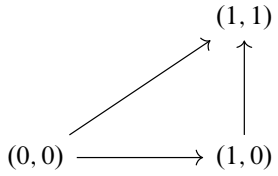


(c) $\mathbb{2} \times \mathbb{2} \times \mathbb{2}$

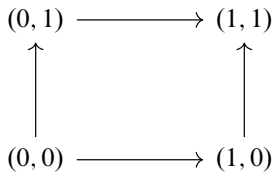
Figure 1: First three powers of $\mathbb{2}$



(a) $\Delta^2 :\equiv \{\langle t, s\rangle : \mathbb{2} \times \mathbb{2} \mid s \leq t\}$



(b) $\partial\Delta^2 :\equiv \{\langle t, s\rangle : \mathbb{2} \times \mathbb{2} \mid (0 \equiv s) \vee (s \equiv t) \vee (t \equiv 1)\}$



(c) $\partial\square :\equiv \{\langle t, s\rangle : \mathbb{2} \times \mathbb{2} \mid (t \equiv 0) \vee (s \equiv 1) \vee (s \equiv 0) \vee (t \equiv 1)\}$

Figure 2: Some important shapes

strict partial order with bottom element 0 and top element 1 [KRW23]. The tope layer allows us to talk about polytopes lying in cubes. That is, the tope layer is a two layer type theory that includes an extra, separate context for cubes. Judgements in the tope layer are generally expressed in the form

$$\Xi \mid \Phi \vdash \phi$$

with judgements

$$\Xi \vdash \top\,\mathsf{tope} \qquad \Xi \mid \Phi \vdash \top \qquad \Xi \vdash \bot\,\mathsf{tope}$$

ensuring that the tope layer forms a bounded lattice. The full formal specification for the rules can be found in fig. 3 and fig. 4.

The tope layer also admits operations of finite disjunction, conjunction, equality, and inequality. The operations in the tope layer allow us to talk about shapes as seen as in fig. 2. Moreover, there is a principle of explosion available to us arising from the tope layer. For all types A, there is a term $\mathsf{rec}_\bot : A$. If the tope layer entails the empty tope, then for all $a : A$, we have that $\mathsf{rec}_\bot \equiv a$. Disjunction allows us to have disjunct terms resulting from disjunct shapes. For all types A that have two shapes ϕ and ψ along with

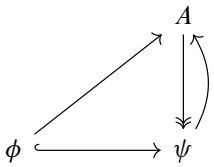
terms depending on those shapes $a_\phi : A$ and $a_\psi : A$ that agree on their intersection, there is a term $\mathsf{rec}_\vee^{\phi,\psi}(a_\phi, a_\psi)$. Whenever the tope context entails ϕ, then $\mathsf{rec}_\vee^{\phi,\psi}(a_\phi, a_\psi) \equiv a_\phi$. On the other hand, whenever the cope context entails ψ, then $\mathsf{rec}_\vee^{\phi,\psi}(a_\phi, a_\psi) \equiv a_\psi$. The formal rules dictating these operations are detailed in fig. 5.

2.2 Extension types

Extension types are important as they allow an easy interface to have types that depend on shapes and *cofibrations*. To say more, suppose we have two shapes ϕ and ψ and a judgement $t : I \mid \phi \vdash \psi$. The judgement can intuitively be viewed as a shape inclusion, and we call this shape inclusion a cofibration. Moreover, given judgements $\psi \mid \Gamma \vdash A$, and $\phi \mid \Gamma \vdash a : A$, we write the type of extensions of a as

$$\left\langle \prod_{t:I\mid\psi} A \middle| {\phi \atop a} \right\rangle \tag{1}$$

An extension type encodes the following diagram:



Remark. For readability purposes, we note that the full formal specification of extension types can be found at fig. 7 and we adopt the following conventions:

1. When A does not depend on a shape, instead of $\left\langle \prod_{t:I\mid\psi} A \middle| {\phi \atop a} \right\rangle$ we write $\left\langle \{t : I \mid \psi\} \to A \middle| {\phi \atop a} \right\rangle$.

2. When ϕ is $\bot$, we observe that a is $\mathsf{rec}_\bot$, so instead of

   $$\left\langle \prod_{t:I\mid\psi} A \middle| {\bot \atop \mathsf{rec}_\bot} \right\rangle,$$

   we write

   $$\prod_{t:I\mid\psi} A$$

   and instead of

   $$\left\langle \{t : I \mid \psi\} \to A \middle|^{\bot}_{\mathsf{rec}_\bot} \right\rangle,$$

   we write

   $$\{t : I \mid \psi\} \to A.$$

3. Since we have extension types defined, we can think of A literally as a function

   $$A : \{t : I \mid \psi\} \to \mathcal{U}$$

with a section $a : \prod_{(t:I \mid \phi)} A(t)$ and write its extension type as

$$\left\langle \prod_{t:I \mid \psi} A(t) \middle| {}^{\phi}_{a} \right\rangle.$$

The syntax of an extension type is not just suggestive. They are intended to be a generalization of dependent functions and as a result we can prove similar facts about extension types as we can about functions. To start, we know that the arguments of a function commute. That is,

$$(X \to Y) \to Z \simeq Y \to (X \to Z)$$

and

$$\prod_{(x:X)} \prod_{(y:Y)} Z(x,y) \simeq \prod_{(y:Y)} \prod_{(x:X)} Z(x,y).$$

The commutativity of the arugments of an extension type also follow.

Lemma 2.2.1 ([RS17], 4.1)**.** *If* $t : I \mid \phi \vdash \psi$ *and* $X : \mathcal{U}$*, while* $Y : \{t : I \mid \phi\} \to X \to \mathcal{U}$ *and* $f : \prod_{(t:I \mid \phi)} \prod_{(x:X)} Y(t,x)$ *then*

$$\left\langle \prod_{t:I \mid \psi} \prod_{(x:X)} Y(t,x) \middle| {}^{\phi}_{f} \right\rangle \simeq \prod_{x:X} \left\langle \prod_{t:I \mid \psi} Y(t,x) \middle| {}^{\phi}_{f} \right\rangle$$

Similarly, we can curry the arguments of a function. That is, there is an equivalence

$$X \to Y \to Z \simeq (X \times Y) \to Z.$$

Extension types also behave similarly, allowing for currying and swapping in multiple argument functions.

Lemma 2.2.2 ([RS17], 4.2)**.** *If* $t : I \mid \phi \vdash \psi$ *and* $s : J \mid \chi \vdash \zeta$*, while*

$$X : \{t : I \mid \psi\} \to \{s : J \mid \zeta\} \to \mathcal{U}$$

and $f : \prod_{(\langle t,s \rangle : I \times J \mid (\phi \wedge \zeta) \vee (\psi \wedge \chi))} X(t,s)$*, then*

$$\begin{aligned}
&\left\langle \prod_{t:I \mid \psi} \left\langle \prod_{s:J \mid \zeta} X(t,s) \middle| {}^{\chi}_{\lambda s.\, f\langle t,s \rangle} \right\rangle \middle| {}^{\phi}_{\lambda t.\, f\langle t,s \rangle} \right\rangle \\
&\qquad \simeq \left\langle \prod_{\langle t,s \rangle : I \times J \mid \psi \wedge \zeta} X(t,s) \middle| {}^{(\phi \wedge \zeta) \vee (\psi \wedge \chi)}_{f} \right\rangle \\
&\qquad \simeq \left\langle \prod_{s:J \mid \zeta} \left\langle \prod_{t:I \mid \psi} X(t,s) \middle| {}^{\phi}_{\lambda s.\, f\langle t,s \rangle} \right\rangle \middle| {}^{\chi}_{\lambda s.\, \lambda t.\, f\langle ts \rangle} \right\rangle
\end{aligned}$$

We can also extend the type theoretic axiom of choice, which states that there is an equivalence

$$\prod_{(x:X)} \sum_{(y:Y(x))} Z(x,y) \simeq \sum_{(f:\prod_{(x:X)} Y(x))} \prod_{(x:X)} Z(x, f(x))$$

to extension types:

Lemma 2.2.3 ([RS17], 4.3)**.** *If $t : I \mid \phi \vdash \psi$, while $X : \{t : I \mid \psi\} \to \mathcal{U}$ and $Y : \prod_{(t:I \mid \psi)}(X(t) \to \mathcal{U})$, while $a : \prod_{(t:I \mid \phi)} X(t)$ and $b : \prod_{(t:I \mid \phi)} Y(t, a(t))$, then*

$$\begin{aligned}&\left\langle \prod_{t:I \mid \psi} \sum_{(x:X(t))} Y(t,x) \middle|^{\phi}_{\lambda t.\,(a(t),b(t))} \right\rangle \\ &\simeq \sum_{f:\left\langle \prod_{t:I \mid \psi} X(t) \middle|^{\phi}_{a} \right\rangle} \left\langle \prod_{t:I \mid \psi} Y(t, f(t)) \middle|^{\phi}_{b} \right\rangle.\end{aligned}$$

While the following two lemmas do not have direct analogus with the usual function type, they are nonetheless extremely useful:

Lemma 2.2.4 ([RS17], 4.4)**.** *Suppose $t : I \mid \phi \vdash \psi$ and $t : I \mid \psi \vdash \chi$, and that $X : \{t : I \mid \chi\} \to \mathcal{U}$ and $a : \prod_{(t:I \mid \phi)} X(t)$. Then,*

$$\left\langle \prod_{t:I \mid \chi} X \middle|^{\phi}_{a} \right\rangle \simeq \sum_{f:\left\langle \prod_{t:I \mid \psi} X \middle|^{\phi}_{a} \right\rangle} \left\langle \prod_{t:I \mid \chi} X \middle|^{\psi}_{f} \right\rangle$$

Lemma 2.2.5 ([RS17], 4.5)**.** *Suppose $t : I \vdash \phi$ and $t : I \vdash \psi$ and that $X : \{t : I \mid \phi \vee \psi\} \to \mathcal{U}$ and $a : \prod_{(t:I \mid \psi)} X(t)$. Then,*

$$\left\langle \prod_{t:I \mid \phi \vee \psi} X \middle|^{\psi}_{a} \right\rangle \simeq \left\langle \prod_{t:I \mid \phi} X \middle|^{\phi \wedge \psi}_{\lambda t.\, a(t)} \right\rangle$$

In HoTT, the relative function extensionality axiom allows us to conclude that $\prod_{(x:X)} B(x)$ is contractible if each $B(x)$ is contractible. We can state a similar axiom using extension types.

Axiom 2.2.6 ([RS17], relative function extensionality, 4.6)**.** *Supposing $t : I \mid \phi \vdash \psi$ and that $A : \{t : I \mid \phi\} \to \mathcal{U}$ is such that each $A(t)$ is contractible, and moreover $a : \prod_{(t:I \mid \phi)} A(t)$, then $\left\langle \prod_{t:I \mid \psi} A(t) \middle|^{\phi}_{a} \right\rangle$ is contractible.*

We can also show that function extensionality follows from axiom theorem 2.2.6.

Lemma 2.2.7 ([RS17], 4.8)**.** *Assuming axiom theorem 2.2.6, given a family $A : \{t : I \mid \psi\}$ and $a : \prod_{(t:I \mid \phi)} A(t)$, and $f, g : \left\langle \prod_{t:I \mid \psi} A(t) \middle|^{\phi}_{a} \right\rangle$, the map*

$$(f = g) \to \left\langle \prod_{t:I \mid \psi} f(t) = g(t) \middle|^{\phi}_{\lambda t.\,\mathsf{refl}} \right\rangle$$

is an equivalence.

As is done in [RS17], we assume axiom theorem 2.2.6.

2.3 Segal condition

Having established our three-layer type theory and extension types, we can start developing the core aspects of sHoTT. We initiate this development by utilizing our extension types to investigate the presence of simplicial structures within types.

Definition 2.3.1 ([RS17], 5.1)**.** For a type A and points $x, y : A$ which induce a term $[x, y]$[§] in context $\partial\Delta^1$, we write

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

and say that an element of $\mathsf{hom}_A(x, y)$ is a **morphism** from x to y.

Viewing Segal types as categories, it is also important to have an object map.

Definition 2.3.2. There are maps:

- $\mathsf{Ob} :\equiv \lambda(A : \mathcal{U}).\, \{t : \Delta^0 \mid \top\} \to A$
- $U : \sum_{(A:\{t:\Delta^0 \mid \top\}\to\mathcal{U})} \mathsf{Ob}(A(\star))$ defined by

$$U(A) :\equiv \lambda(t : \Delta^0).\, \lambda(a : A(t)).\, a$$

- For any type A, $\iota : \mathsf{Ob}(A) \to A$ defined by

$$\iota a :\equiv a(\star)$$

Before we probe for more shapes, we turn our attention to the functorality of functions in sHoTT. In the last section, we saw that functions acted functorially on paths using path induction. In sHoTT, functions act functorially on morphisms by definition. Consequently, we have the following lemmas.

Lemma 2.3.3 ([RS17], 6.1)**.** *Let A, B be types and $f : A \to B$. Then, given $x, y : A$, there is a function*

$$\mathsf{map}_f : \mathsf{hom}_A(x, y) \to \mathsf{hom}_B(fx, fy)$$

When the context is clear, we often write $f(h)$ rather than $\mathsf{map}_f h$ for $h : \mathsf{hom}_A(x, y)$.

Our exploration extends beyond one dimension. We can also probe for 2-dimensional shapes and higher. For the purposes of this thesis, we focus on 1 and 2-dimensional shapes, and we will not delve into higher dimensional definitions here.

Definition 2.3.4 ([RS17], 5.2)**.** For points $x, y, z : A$ and morphisms $f : \mathsf{hom}_A(x, y)$, $g : \mathsf{hom}_A(y, z)$, and $h : \mathsf{hom}_A(x, z)$ which induces a term $[x, y, z, f, g, h]$ in context $\partial\Delta^2$, we write

$$\hom^2_A\left(\begin{array}{ccccc} & & y & & \\ & f \nearrow & & \searrow^{g} & \\ x & & \xrightarrow[h]{} & & z \end{array} \right) :\equiv \left\langle \Delta^2 \to A \middle|^{\partial\Delta^2}_{[x,y,z,f,g,h]} \right\rangle$$

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**.

[§]This is shorthand for $\mathsf{rec}^{t\equiv 0, t\equiv 1}_{\vee}(x, y)$

Definition 2.3.5. For points $a, b, c, d : A$, and morphisms $f : \mathsf{hom}_A(a, d)$, $g : \mathsf{hom}_A(b, c)$, $h : \mathsf{hom}_A(a, b)$, and $k : \mathsf{hom}_A(d, c)$ which induces a term $[a, b, c, d, f, k, h, g]$ in context $\partial\square$, we write

$$\mathsf{diag}^2_A\left(\begin{array}{ccc} d & \xrightarrow{\quad k \quad} & c \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ a & \xrightarrow[\quad h \quad]{} & b \end{array}\right) :\equiv \left\langle \mathbb{2} \times \mathbb{2} \to A \middle|^{\partial\square}_{[a,b,c,d,f,k,h,g]} \right\rangle$$

or, in an abbreviated fashion,

$$\mathsf{diag}^2_A(f, k; h, g)$$

and say that an element of $\mathsf{diag}^2_A(f, k; h, g)$ is a **diagram**.

Importantly, $\mathbb{2} \times \mathbb{2}$ is the pushout of two copies of Δ^2 along their common boundary. We can use this fact in two ways. The first allows us to create "connection squares", which are simple but powerful diagrams:

Lemma 2.3.6 ([RS17], 3.5)**.** *For any type A, points $a, b : A$, and morphism $f : \mathsf{hom}_A(a, b)$, there are diagrams*

$$\bigvee\nolimits_f : \mathsf{diag}^2_A\left(\begin{array}{ccc} y & \xrightarrow{\quad \mathsf{idhom}_y \quad} & y \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \mathsf{idhom}_y} \\ x & \xrightarrow[\quad f \quad]{} & y \end{array}\right)$$

$$\bigwedge\nolimits_f : \mathsf{diag}^2_A\left(\begin{array}{ccc} x & \xrightarrow{\quad f \quad} & y \\ {\scriptstyle \mathsf{idhom}_x}\downarrow & & \downarrow{\scriptstyle f} \\ x & \xrightarrow[\quad \mathsf{idhom}_x \quad]{} & x \end{array}\right)$$

In the second, we note that diagrams inherit the pushout property. More specifically, the following lemma shows that diagrams are a pushout of two 2-morphisms.

Lemma 2.3.7. *For any type A and morphisms $f : \mathsf{hom}_A(a, d)$, $g : \mathsf{hom}_A(b, c)$, $h : \mathsf{hom}_A(a, b)$, and $k : \mathsf{hom}_A(d, c)$, there is an equivalence*

$$\mathsf{diag}^2_A(f, k; h, g) \simeq \sum_{p:\mathsf{hom}_A(a,c)} \hom^2_A(f, k; p) \times \hom^2_A(h, g; p) \tag{2}$$

Proof. From left to right,

$$\alpha \mapsto \left(\lambda(t:\mathbb{2}).\,\alpha(t,t), \lambda(\langle t,s\rangle:\Delta^2).\,\alpha(s,t), \lambda(\langle t,s\rangle:\Delta^2).\,\alpha(t,s)\right).$$

From right to left,

$$(p,\alpha_1,\alpha_2) \mapsto \lambda(\langle t,s\rangle :\mathbb{2}\times\mathbb{2}).\,\mathsf{rec}_\vee^{t\le s, s\le t}(\alpha_1(s,t),\alpha_2(t,s)).$$

We first check, in the case of the map from the right to the left, that the two provided terms for $\mathsf{rec}_\vee$ agree along the common boundary. Since $\alpha_1(t,t) \equiv \alpha_2(t,t) \equiv p$ for all $t:\mathbb{2}$, the map is well-formed.

The round trip from left, to right, then back to the left takes a term

$$\alpha : \left\langle \mathbb{2}\times\mathbb{2} \to A \middle|^{\partial\square}_{[f,k,h,g]} \right\rangle$$

to

$$\left(\lambda(t:\mathbb{2}).\,\alpha(t,t), \lambda(\langle t,s\rangle:\Delta^2).\,\alpha(s,t), \lambda(\langle t,s\rangle:\Delta^2).\,\alpha(t,s)\right)$$

then finally to

$$\lambda(\langle t,s\rangle:\mathbb{2}\times\mathbb{2}).\,\mathsf{rec}_\vee^{t\le s, s\le t}(\alpha(t,s),\alpha(t,s))$$

which is definitionally equal to α.

The round starting from the right takes the term

$$(p,\alpha_1,\alpha_2) : \sum_{p:\mathsf{hom}_A(a,c)} \hom_A^2(f,k;p)\times\hom_A^2(h,g;p)$$

to

$$\lambda(\langle t,s\rangle :\mathbb{2}\times\mathbb{2}).\,\mathsf{rec}_\vee^{t\le s, s\le t}(\alpha_1(s,t),\alpha_2(t,s)).$$

We can evaluate what the map from left to right does on this term coordinate-wise. In the first coordinate, we have

$$\lambda(t:\mathbb{2}).\,(\lambda(\langle t,s\rangle :\mathbb{2}\times\mathbb{2}).\,\mathsf{rec}_\vee^{t\le s, s\le t}(\alpha_1(s,t),\alpha_2(t,s)))(t,t) \equiv p.$$

In the second coordinate, we have the following term in the context of $\langle t,s\rangle : \Delta^2$:

$$(\lambda(\langle t,s\rangle :\mathbb{2}\times\mathbb{2}).\,\mathsf{rec}_\vee^{t\le s, s\le t}(\alpha_1(s,t),\alpha_2(t,s)))(s,t) \tag{3}$$

Noting that $s \le t$, equation eq. (3) computes to $\alpha_1(t,s)$. Thus, the second coordinate is simply α_1.

In the third coordinate, we have the following term in the context of $\langle t,s\rangle : \Delta^2$:

$$(\lambda(\langle t,s\rangle :\mathbb{2}\times\mathbb{2}).\,\mathsf{rec}_\vee^{t\le s, s\le t}(\alpha_1(s,t),\alpha_2(t,s)))(t,s) \tag{4}$$

Again, since $s \le t$, equation eq. (4) computes to $\alpha_2(t,s)$, thus the third coordinate is simply α_2.

Since both round trips are definitionally the identity, the equivalence in eq. (2) holds. □

Observe that, although we have introduced categorical terms such as diagrams and morphisms, a formal definition of a category in sHoTT is still forthcoming. Recall that in HoTT, path composition was a natural consequence of path induction, subsequently revealing the groupoidal nature of types. While it is tempting to anticipate a similar derivation for morphism composition, this expectation does not hold. Morphism composition in sHoTT requires a different approach, as it is not a direct result of inductive principles as with paths.

This distinction highlights a core principle of sHoTT: that types exhibit the structure of infinity categories. To illustrate this concept through an analogy with programming languages, path composition mirrors the behavior of a parametrically polymorphic function, where the operation is uniform across all types. In contrast, morphism composition is analogous to type class polymorphism, where the implementation of composition can vary depending on the specific type.

It is not the case that every type naturally exhibits the structure of a category. Rather, categorical structure emerges specifically in those types that admit a well-defined composition operation. Indeed, we can demonstrate that the existence of such a composition operation serves as the foundational property from which all other aspects of categorical structure are derived. Consequently, we proceed to our next definition.

Definition 2.3.8 ([RS17], 5.3)**.** Let A be a type, $x, y, z : A$ and $f : \mathsf{hom}_A(x, y)$ and $g : \mathsf{hom}_A(y, z)$. We say A is **Segal** if the type

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

is contractible. The unique inhabitant is denoted by $(g \cdot f, \circ_{gf})$.

Thus, in Segal types, composable morphisms directly correspond to 2-mor phisms. Before detailing the categorical structure of Segal types, we will examine alternative characterizations and key lemmas. We begin with an alternative characterization.

Lemma 2.3.9 ([RS17], 5.5)**.** *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.

In the proof of theorem 2.3.9, [RS17] proves along the way that there is an equivalence

$$\sum_{h:\mathsf{hom}_A(x,z)} \hom^2_A \left(\begin{array}{ccccc} & & y & & \\ & f \swarrow & & \searrow g & \\ x & & \xrightarrow[h]{} & & z \end{array} \right) \simeq \left\langle \Delta^2 \to A \middle|^{\Lambda^2_1}_{[x,y,z,f,g]} \right\rangle. \tag{5}$$

As we will see, it is also useful to consider summing over the other sides of a 2-morphism.

Proposition 2.3.10. *In any type A, there are equivalences*

$$\sum_{f:\hom_A(x,y)} \hom_A^2\left(\begin{array}{ccccc} & & y & & \\ & f\nearrow & & \searrow g & \\ x & & \xrightarrow[h]{} & & z \end{array}\right) \simeq \left\langle \Delta^2 \to A \middle|^{\Lambda^2_2}_{[x,y,z,g,h]} \right\rangle \tag{6}$$

and

$$\sum_{g:\hom_A(y,z)} \hom_A^2\left(\begin{array}{ccccc} & & y & & \\ & f\nearrow & & \searrow g & \\ x & & \xrightarrow[h]{} & & z \end{array}\right) \simeq \left\langle \Delta^2 \to A \middle|^{\Lambda^2_0}_{[x,y,z,f,h]} \right\rangle. \tag{7}$$

Proof. Let Δ^1_2 denote the bottom leg and Δ^1_0 denote the right leg of Δ^2. We can observe then that

- $\Lambda^2_2 \cap \Delta^1_2 = \partial\Delta^1_2$ and $\Lambda^2_0 \cap \Delta^1_0 = \partial\Delta^1_0$
- $\Lambda^2_2 \cup \Delta^1_2 = \partial\Delta^2$ and $\Lambda^2_0 \cup \Delta^1_0 = \partial\Delta^2$.

Hence, by theorem 2.2.5, there are equivalences

$$\begin{aligned} \sum_{f:\hom_A(x,y)} \hom_A^2\left(\begin{array}{ccccc} & & y & & \\ & f\nearrow & & \searrow g & \\ x & & \xrightarrow[h]{} & & z \end{array}\right) \equiv \\ \sum_{f:\left\langle \Delta^1 \to A \middle|^{\partial\Delta^1}_{[x,y]} \right\rangle} \left\langle \Delta^2 \to A \middle|^{\partial\Delta^2}_{[x,y,z,f,g,h]} \right\rangle \simeq \\ \sum_{l:\left\langle \partial\Delta^2 \to A \middle|^{\Lambda^2_2}_{[x,y,z,g,h]} \right\rangle} \left\langle \Delta^2 \to A \middle|^{\partial\Delta^2}_{l} \right\rangle \simeq \\ \left\langle \Delta^2 \to A \middle|^{\Lambda^2_2}_{[x,y,z,g,h]} \right\rangle \end{aligned}$$

and

$$\begin{aligned} \sum_{g:\hom_A(y,z)} \hom_A^2\left(\begin{array}{ccccc} & & y & & \\ & f\nearrow & & \searrow g & \\ x & & \xrightarrow[h]{} & & z \end{array}\right) \equiv \\ \sum_{g:\left\langle \Delta^1 \to A \middle|^{\partial\Delta^1}_{[y,z]} \right\rangle} \left\langle \Delta^2 \to A \middle|^{\partial\Delta^2}_{[x,y,z,f,g,h]} \right\rangle \simeq \\ \sum_{l:\left\langle \partial\Delta^2 \to A \middle|^{\Lambda^2_0}_{[x,y,z,f,h]} \right\rangle} \left\langle \Delta^2 \to A \middle|^{\partial\Delta^2}_{l} \right\rangle \simeq \\ \left\langle \Delta^2 \to A \middle|^{\Lambda^2_0}_{[x,y,z,f,h]} \right\rangle \end{aligned}$$

with, in both cases, the first equivalence following from theorem 2.2.5 and the second theorem 2.2.4. □

Now that we've covered that digression, we can examine the categorical structure of Segal types. We already have three key components: objects (which are points of a type), morphisms (which are points of the morphism type), and a weak composition function that arises from the Segal definition. Unit and associativity laws also follow.

Lemma 2.3.11 ([RS17], 5.8)**.** *If A is a Segal type with terms $x, y : A$, then for any $f : \mathsf{hom}_A(x, y)$ we have $\mathsf{idhom}_y \circ f = f$ and $f \circ \mathsf{idhom}_x = f$.*

Lemma 2.3.12 ([RS17], 5.9)**.** *If A is a Segal type with terms $x, y, z, w : A$, then for any $f : \mathsf{hom}_A(x, y)$, $g : \mathsf{hom}_A(y, z)$, $h : \mathsf{hom}_A(z, w)$ we have $(h \circ g) \circ f = h \circ (g \circ f)$.*

We turn now to the analogue of homotopies in sHoTT. Given a Segal type A, we have two ways of saying two morphisms $f, g : \mathsf{hom}_A(x, y)$ are the same. We can either:

1. Provide a path $f = g$
2. Provide a 2-simplex $\hom^2_A\left(\begin{array}{ccccc} & & x & & f \\ & \mathsf{idhom}_x \nearrow & & \searrow & \\ x & & \xrightarrow[g]{} & & y \end{array}\right)$

These two perspectives turn out to be equivalent.

Lemma 2.3.13 ([RS17], 5.10)**.** *For any $f, g : \mathsf{hom}_A(x, y)$ in a Segal type A, the natural map*

$$f = g \to \hom^2_A\left(\begin{array}{ccccc} & & x & & f \\ & \mathsf{idhom}_x \nearrow & & \searrow & \\ x & & \xrightarrow[g]{} & & y \end{array}\right)$$

is an equivalence.

Furthermore, composition and equality are closely related.

Lemma 2.3.14 ([RS17], 5.12)**.** *For any $f : \mathsf{hom}_A(x, y)$, $g : \mathsf{hom}_A(y, z)$, and $h : \mathsf{hom}_A(x, z)$ in a Segal type A, the natural map*

$$g \cdot f = h \to \hom^2_A\left(\begin{array}{ccccc} & & y & & g \\ & f \nearrow & & \searrow & \\ x & & \xrightarrow[h]{} & & z \end{array}\right)$$

is an equivalence.

Extending this idea, commutative squares correspond to commuting morphisms.

Lemma 2.3.15. *Given a Segal type A and compatible morphisms $f : \mathsf{hom}_A(a, d)$, $k : \mathsf{hom}_A(d, c)$, $h : \mathsf{hom}_A(a, b)$, and $g : \mathsf{hom}_A(b, c)$ then*

$$\mathsf{diag}^2_A\left(\begin{array}{ccc} d & \xrightarrow{k} & c \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ a & \xrightarrow[h]{} & b \end{array}\right) \simeq k \cdot f = g \cdot h \tag{8}$$

Proof. By lemma theorem 2.3.7,

$$\mathsf{diag}^2_A(f,k;h,g) \simeq \sum_{p:\mathsf{hom}_A(a,c)} \hom^2_A(f,k;p) \times \hom^2_A(h,g;p).$$

By lemma theorem 2.3.15, the both 2-morphism types in the codomain can be equated with path types

$$\sum_{p:\mathsf{hom}_A(a,c)} \hom^2_A(f,k;p) \times \hom^2_A(h,g;p) \simeq \sum_{p:\mathsf{hom}_A(a,c)} k \cdot f = p \times g \cdot h = p.$$

By lemma theorem 1.8.2, we can reassociate the pair type as follows

$$\sum_{p:\mathsf{hom}_A(a,c)} k \cdot f = p \times g \cdot h = p \simeq \sum_{t:\sum_{(p:\mathsf{hom}_A(a,c))} k\cdot f = p} g \cdot h = \mathsf{fst}(t).$$

Note that the base is contractible by singleton contractibility, with center of contraction $k \cdot f$. Thus, by lemma theorem 1.5.8, the codomain is equivalent to

$$g \cdot h = k \cdot f.$$

Our desired equation is the composition of all of the equivalences constructed in this proof. □

2.4 Discrete types

Rather than probing for categorical structure, we can instead probe for groupoid structure. Specifically, a type that has no non-trivial simplicial data is a groupoid. This is captured in the following definition.

Definition 2.4.1 ([RS17], 7.1)**.** For any type A, there is a map

$$\mathsf{idtoarr} : \prod_{x,y:A} x =_A y \to \mathsf{hom}_A(x,y) \tag{9}$$

defined by path induction and the equation $\mathsf{idtoarr}(\mathsf{refl}_x) :\equiv \mathsf{idhom}_x$. We say A is a **discrete type** if $\mathsf{idtoarr}_{x,y}$ is an equivalence for all $x, y : A$.

We can show that discrete types have the structure of a category.

Lemma 2.4.2 ([RS17], 7.3)**.** *If A is a discrete type them A is Segal.*

2.5 Rezk types

The astute reader will have noticed that having defined Segal types, we have two competing notions of equality. There is a tension between isomorphisms between objects and paths between objects, which causes poor behavior. We are then interested in types where these two notions coincide. This motivates the next type class definition. First, we must make precise what it means to be an isomorphism.

Definition 2.5.1 ([RS17], 10.1)**.** For any Segal type A and $f : \mathsf{hom}_A(x, y)$, we write

$$\mathsf{isiso}(f) :\equiv \left(\sum_{g:\mathsf{hom}_A(y,x)} g \cdot f = \mathsf{idhom}_x\right) \times \left(\sum_{h:\mathsf{hom}_A(y,x)} f \cdot h = \mathsf{idhom}_y\right)$$

and say that f is an **isomorphism** if $\mathsf{isiso}(f)$ is inhabited. Similarly, for any $x, y : A$, we write **the type of isomorphisms** from x to y as

$$x \cong y :\equiv \sum_{f:\mathsf{hom}(A,x)y} \mathsf{isiso}(f)$$

This coincides with our notion of a bi-invertible map. Just like in HoTT, we can show that inverses and bi-invertible maps are logically equivalent.

Lemma 2.5.2 ([RS17], 10.1)**.** *For any Segal type A and morphism $f : \mathsf{hom}_A(x, y)$, f is an isomorphism if and only if there is a $g : \mathsf{hom}_A(y, x)$ such that $g \cdot f = \mathsf{idhom}_x$ and $f \cdot g = \mathsf{idhom}_y$.*

We can also show that our type of isomorphisms is a proposition.

Lemma 2.5.3 ([RS17], 10.2)**.** *For any Segal type A and morphism $f : \mathsf{hom}_A(x, y)$, the type $\mathsf{isiso}(f)$ is a proposition.*

Now, we can start to reduce the tension between isomorphisms and identities. First, we can relate them by a map.

Lemma 2.5.4 ([RS17], 10.5)**.**

$$\mathsf{idtoiso} : \prod_{x,y:A} x =_A y \to x \cong_A y \tag{10}$$

*defined by path induction and the equation $\mathsf{idtoiso}(\mathsf{refl}_x) :\equiv \mathsf{idhom}_x$. If A is a Segal type and $\mathsf{idtoiso}_{x,y}$ is an equivalence for all x, y, we say A is **Rezk-complete** and that A is a **Rezk type**.*

2.6 Covariant fibrations

do codes for sigma and pi

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. When discussing which type families lift simplicial shapes, we must specify which shapes are being lifted. An important collection of type families are the ones that lift *morphisms*, covariantly or contravariantly. Before we can precisely define such type families, we must first specify what it means to be a "lifted morphism".

Definition 2.6.1 ([RS17], 8.1)**.** Given a type A, a type family $C : A \to \mathcal{U}$, points $x, y : A$, a morphism $f : \mathsf{hom}_A(x, y)$ and points $u : C(x)$ and $v : C(y)$, the type of **dependent morphisms over f from u** to v is written as

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

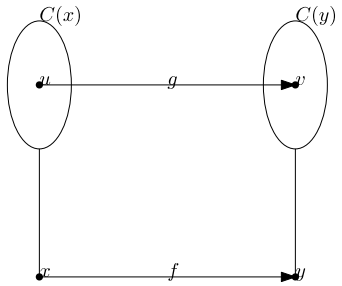
With the dependent morphism defined, we can show that dependent functions can also be definitionally applied to morphisms.

Lemma 2.6.2. *Let $B : A \to \mathcal{U}$ and $\phi : \prod_{(x:A)} B(x)$. Then, given $x, y : A$, there is a function*

$$\mathsf{mapd}_{\phi} : \prod_{f:\mathsf{hom}_A(x,y)} \mathsf{hom}_{B(f)}\left(\phi(x), \phi(y)\right)$$

Proof. Same as theorem 2.3.3. □

The dependent morphism represents the intuitive notion of "morphism in the total space". It is instructive to view an inhabitant $g : \mathrm{hom}_{C(f)}(u, v)$ as literally living above the morphism f, as the following diagram illustrates:



We do not know without extra information that C depends functorially on A. So, we can instead restrict ourselves to the study of type families that satisfy do act as covariant functors.

Definition 2.6.3 ([RS17], 8.2)**.** For any type A, a type family $C : A \to \mathcal{U}$ is **covariant** if for every $f : \mathsf{hom}_A(x, y)$ and $u : C(x)$, the type

$$\sum_{v:C(y)} \mathrm{hom}_{C(f)}(u, v)$$

is contractible. When the context is clear, we write the specific inhabitant as $(\mathsf{dtransport}^C(f, u), \mathsf{trans}_{f,u})$ or even more tersely as $(f_* u, \mathsf{trans}_{f,u})$.

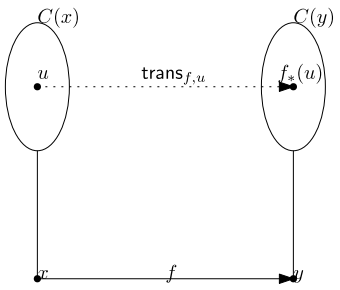
Dually[¶], C is **contravariant** if for every $f : \mathsf{hom}_A(x, y)$ and $v : C(y)$, the type

$$\sum_{v:C(x)} \mathrm{hom}_{C(f)}(u, v)$$

is contractible.

We can also alternatively, but perhaps more traditionally, characterize covariant fibrations as type families that satisfy a lifting property. A covariant fibration not only lifts paths, but lifts them uniquely. The following diagram encapsulates this:

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



The dotted line represents the uniqueness of $\mathsf{trans}_{f,u}$ as a morphism starting at u living over f. The diagram is thus formalized by the following lemma.

Lemma 2.6.4 ([RS17], 8.4)**.** *A type family $C : A \to \mathcal{U}$ is covariant if and only if for all morphisms $f : \mathsf{hom}_A(x, y)$ and points $u : C(x)$ the type $\left\langle \prod_{t:\Delta^1} C(f(t)) \middle|^{0}_{u} \right\rangle$ is contractible; there is a unique lifting of f that starts at u.*

Remark. Covariant families are stable under substitution. That is, for any $g : B \to A$ and covariant type family $C : A \to \mathcal{U}$, the type family $\lambda b.\, C(g(b))$ is also covariant.

Having explored various characterizations of the covariant property, we now focus on the properties specific to our covariant type families. To begin, in HoTT, transporting along a type family is equivalent to applying the type family to a path and then coercing along the resulting path. The directed version of the result holds in sHoTT.

Lemma 2.6.5. *Let $C : B \to \mathcal{U}$ be covariant and $h : A \to B$. For any $f : \mathsf{hom}_A(x, y)$ and $u : C(h(x))$ we have that*

$$\mathsf{dtransport}^{C \circ h}(f, u) = \mathsf{dtransport}^{C}(\mathsf{map}_h f, u)$$

Proof. Note that

$$(\mathsf{dtransport}^{C \circ h}(f, u), \mathsf{trans}_{f,u}) : \sum_{v : C(h(y))} \mathsf{hom}_{(C \circ h)(f)}(u, v) \equiv$$

and

$$(\mathsf{dtransport}^{C}(\mathsf{map}_h f, u), \mathsf{trans}_{\mathsf{map}_h f, u}) : \sum_{v : C(h(y))} \mathsf{hom}_{C(\mathsf{map}_h f)}(u, v) \equiv \sum_{v : C(h(y))} \mathsf{hom}_{C(h(f))}(u, v)$$

Since $C \circ h$ is covariant, the type $\sum_{(v : C(h(y)))} \mathsf{hom}_{C(h(f))}(u, v)$ is contractible, hence $\mathsf{dtransport}^{C \circ h}(f, u) = \mathsf{dtransport}^{C}(\mathsf{map}_h f, u)$, as desired. □

We can also show that dependent morphisms in a covariant type family from some are equivalent to a path whose domain is a transport of the source of the dependent morphism, analogous to the situation in HoTT.

Lemma 2.6.6 ([RS17], 8.15)**.** *If $C : A \to \mathcal{U}$ is covariant, $f : \mathsf{hom}_A(x, y)$, $u : C(x)$, and $v : C(y)$, then*

$$\mathsf{hom}_{C(f)}(u, v) \simeq (f_* u =_{C(y)} v).$$

We now shift our focus to the categorical nature of fibrations, particularly in the context of covariant families over Segal bases. Such covariant families exhibit an important property: directed transporting is functorial. This property is formalized in the following lemma.

Lemma 2.6.7 ([RS17], 8.16)**.** *For any Segal type A, covariant family $C : A \to \mathcal{U}$, $f : \mathsf{hom}_A(x, y)$, $g : \mathsf{hom}_A(y, z)$, and $u : C(X)$, we have*

$$g_*(f_* u) = (g \cdot f)_* u \qquad \text{and} \qquad (\mathsf{idhom}_x)_* u = u$$

Furthermore, covariant families over Segal bases faithfully represent compositions in the base. This follows from the functoriality of the type family with respect to the base. As a result, the total space of the type family inherits the Segal property, where weak composition in the total space arises naturally from the composition in the base. This is captured in the following lemma:

Lemma 2.6.8 ([RS17], 8.8)**.** *If A is Segal and $C : A \to \mathcal{U}$ is covariant, then*

$$\sum_{a:A} C(a)$$

is Segal.

Notably, certain type families are automatically covariant when defined over a Segal base. A significant example, drawn from category theory, is the representable type family $\mathsf{hom}_A(a, -)$. In sHoTT, the representable plays an equally critical role.

One particularly useful property of the representable in sHoTT is its ability to characterize Segal types. This relationship is formalized in the following lemma:

Lemma 2.6.9 ([RS17], 8.13)**.** *Given a type A and a point $a : A$, then the type family*

$$\lambda x.\, \mathsf{hom}_A(a, x) : A \to \mathcal{U}$$

is covariant if and only if A is Segal.

Covariant type families act functorially in other important ways. For example, naturality comes for free, for maps between covariant types.

Lemma 2.6.10 ([RS17], 8.17)**.** *For any two covariant families $C, D : A \to \mathcal{U}$ and a fiberwise map $\phi : \prod_{(x:A)} C(x) \to D(x)$, $f : \mathsf{hom}_A(x, y)$ and $u : C(x)$,*

$$\phi_y(f_* u) = f_*(\phi_x(u)).$$

An important structural property of covariant families over Segal bases is that their fibers are discrete. This result provides a foundational link between Segal bases and the nature of the types within covariant families:

Lemma 2.6.11 ([RS17], 8.18)**.** *For any Segal type A and covariant map $C : A \to \mathcal{U}$, and $x : A$, the type $C(x)$ is discrete.*

Corollary 2.6.11.1 ([RS17], 8.19)**.** *If A is Segal, then for any $x, y : A$ the type $\mathsf{hom}_A(x, y)$ is discrete.*

Corollary 2.6.11.2 ([RS17], 8.20)**.** *If A is discrete, then for any $x, y : A$, so is $x =_A y$.*

When dealing with multiple argument functions, covariance can be defined:

Definition 2.6.12. Given any type family $P : A \to \mathcal{U}$, a type family $Q : \prod_{(x:A)} P(x) \to \mathcal{U}$ is covariant if the type family

$$C :\equiv \lambda(x, p).\, Q(x, p) : (\sum_{x:A} P(x)) \to \mathcal{U}$$

is covariant.

We can show that the path types in covariant families are covariant:

Lemma 2.6.13 ([RS17], 8.26)**.** *Suppose $C : A \to \mathcal{U}$ is covariant. Then*

$$\lambda a.\, \lambda u.\, \lambda u.\, u = v : \prod_{a:A} (C(a) \to C(a) \to \mathcal{U})$$

is covariant.

The previous lemma is slightly easier to use in the following incarnation:

Corollary 2.6.13.1. *Suppose $C : A \to \mathcal{U}$ is covariant and we have maps $u : \prod_{(x:A)} C(v)$ and $v : \prod_{(x:A)} C(x)$ Then*

$$\lambda x.\, u(x) = v(x) : (A \to \mathcal{U})$$

is covariant.

Proof. Since C is covariant,

$$\lambda a.\, \lambda u.\, \lambda u.\, u = v : \prod_{a:A} (C(a) \to C(a) \to \mathcal{U})$$

is covariant by theorem 2.6.13. So, by theorem 2.6.12,

$$\lambda(x, u, v).\, u = v : (\sum_{x:A} C(x) \times C(x)) \to \mathcal{U}$$

is covariant. We can then precompose with the map $x \mapsto (x, u(x), v(x))$ to form the family

$$\lambda x.\, u(x) = v(x),$$

with covariance of this family coming from our remark above. □

Mapping into a covariant family is covariant:

Lemma 2.6.14 ([RS17], 8.30)**.** *For 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.

Mapping out of a covariant family *into a discrete type* is contravariant:

Lemma 2.6.15 ([RS17], 8.31)**.** *For a discrete type Y and a covariant type family $C : A \to \mathcal{U}$, the type*

$$\lambda(x : A).\, C(a) \to Y$$

is contravariant.

The closure properties also allow us to characterize the action of transport on objects. For example transporting in a pair is, as expected, a pair of transports.

Corollary 2.6.15.1. *Given type families $P : A \to \mathcal{U}$, $Q : \prod_{(x:A)} P(x) \to \mathcal{U}$ that are covariant, a morphism $f : \mathsf{hom}_A(x, y)$, point $p : P(x)$, and map $\phi : \prod_{(p:P(x))} Q(x, p)$,*

$$(f_*(p), (f_* \circ \phi)(p)) = f_*(p, \phi(p))$$

Proof. By theorem 2.6.12, $C :\equiv \lambda x.\, \sum_{(p:P(x))} Q(x, p)$ is covariant. Note that

$$(f_*(p, \phi(p)), \mathsf{trans}_{f,(p,\phi(p))}) : \sum_{v:C(y)} \mathsf{hom}_{C(f)}((p, \phi(p)), v)$$

and similarly

$$(((f_*(p), (f_* \circ \phi)(p))), (\mathsf{trans}_{f,p}, \mathsf{trans}_{f,\phi(p)})) : \sum_{v:C(y)} \mathsf{hom}_{C(f)}((p, \phi(p)), v)\,.$$

The covariance of C implies the type $\sum_{(v:C(y))} \mathsf{hom}_{C(f)}((p, \phi(p)), v)$ is contractible. Hence,

$$f_*(p, \phi(p)) = ((f_*(p), (f_* \circ \phi)(p)))$$

□

Finally, we turn our attention to classifying universes for fibrations. In HoTT, the fact that all type families are fibrations allows us to view $\mathcal{U}$ as a classifying universe for fibrations. On the other hand, in sHoTT only some type families are covariant fibrations. While $\mathcal{U}$ is not a classifying universe for covariant fibrations, we can postulate the existence of a universe that is.

Axiom 2.6.16. *There is a Segal type $\mathcal{U}_{\mathsf{cov}}$ that satisfies the following properties:*

- *There is a covariant function $\mathsf{El} : \mathcal{U}_{\mathsf{cov}} \to \mathcal{U}$.*

- *For any closed discrete type $A : \mathcal{U}$, there is a term $\overline{A} : \mathcal{U}_{\mathsf{cov}}$ such that $\mathsf{El}(\overline{A}) \equiv A$.*
- *For any Segal types $A, B : \mathcal{U}$ satisfying*
  - $i : \mathsf{Ob}(A) \simeq \mathsf{Ob}(B)$
  - $\prod_{(a,b:\mathsf{Ob}A)} \mathsf{hom}_A\,(a,b) \simeq \mathsf{hom}_B\,(i(b), i(a))$

  there is a map
  $$\Phi : A \times B \to \mathcal{U}_{\mathsf{cov}}$$
  such that
  $$\lambda(b:B).\,\mathsf{El}(\Phi(a,b)) \equiv \lambda(b:B).\,\mathsf{hom}_B\,(i(a), b)$$
  $$\lambda(a:A).\,\mathsf{El}(\Phi(a,b)) \equiv \lambda(a:A).\,\mathsf{hom}_B\,(i(a), b)$$
- *Finally, for any $A : \mathcal{U}_{\mathsf{cov}}$ and $P : \mathsf{El}(A) \to \mathcal{U}_{\mathsf{cov}}$, there is a term*
  $$\overline{\Sigma} A\ P : \mathcal{U}_{\mathsf{cov}}$$
  such that
  $$\mathsf{El}(\overline{\Sigma} A\ P) \equiv \sum_{a:\mathsf{El}(A)} \mathsf{El}(P(a))$$

2.7 Yoneda lemma

With the concept of covariant fibrations developed, we are now equipped to delve into a foundational result in category theory: the Yoneda Lemma. The structure of covariant families naturally provides mappings necessary to state the Yoneda Lemma in sHoTT. That is, for any covariant $C : A \to \mathcal{U}$ and $a : A$, there are canonical maps

$$\mathsf{evid}_a^C :\equiv \lambda\phi.\,\phi(a, \mathsf{idhom}_a) : \left(\prod_{x:A} (\mathsf{hom}_A\,(a,x)) \to C(x)\right) \to C(a)$$
$$\mathsf{yon}_a^C :\equiv \lambda\phi.\,\phi(a, \mathsf{idhom}_a) : C(a) \to \left(\prod_{x:A} (\mathsf{hom}_A\,(a,x)) \to C(x)\right)$$

For covariant families defined over Segal types, the Yoneda Lemma is fully realized as follows:

Theorem 2.7.1 ([RS17], 9.1)**.** *Given a covariant family $C : A \to \mathcal{U}$ over a Segal type A, for any $a : A$ the maps evid_A^C and yon_a^C form an equivalence.*

We can also state a dependent version of the Yoneda lemma, which is analogous to a directed version of the path induction principle.

Theorem 2.7.2 ([RS17], 9.5)**.** *For any Segal type A, $a : A$, and covariant type family $C : \prod_{(x:A)}(\mathsf{hom}_A\,(a,x) \to \mathcal{U})$, the function*

$$\mathsf{evid}_a^C\,\lambda\phi.\,\phi(a,\mathsf{idhom}_a) : \left(\prod_{(x:A)}\prod_{(f:\mathsf{hom}_A(a,x))} \to C(x,f)\right) \to C(a,\mathsf{idhom}_a)$$

is an equivalence.

2.8 Directed univalence

For points $A, B : \mathcal{U}_{\mathsf{cov}}$, we seek to make an identification between morphisms $\mathsf{hom}_{\mathcal{U}_{\mathsf{cov}}}\,(A,B)$ and functions between types $\mathsf{El}(A) \to \mathsf{El}(B)$. That is, we seek to state some directed univalence principle that allows us to identify morphisms with functions. To formalize this notion, we start by defining some canonical function:

Lemma 2.8.1. *For any two $A, B : \mathcal{U}_{\mathsf{cov}}$, there is a function*

$$\mathsf{homtofun} : \mathsf{hom}_{\mathcal{U}_{\mathsf{cov}}}\,(A,B) \to (\mathsf{El}(A) \to \mathsf{El}(B)).$$

Proof. We begin by observing that the identity function on $\mathcal{U}_{\mathsf{cov}}$,

$$\mathsf{id}_{\mathcal{U}_{\mathsf{cov}}} : \mathcal{U}_{\mathsf{cov}} \to \mathcal{U}_{\mathsf{cov}}$$

induces a covariant function

$$F :\equiv \lambda(A:\mathcal{U}_{\mathsf{cov}}).\,\mathsf{El}(A) : \mathcal{U}_{\mathsf{cov}} \to \mathcal{U}.$$

So, by Lemma theorem 2.6.14 the type family

$$C :\equiv \lambda(B:\mathcal{U}_{\mathsf{cov}}).\,\mathsf{El}(A) \to F(B) : \mathcal{U}_{\mathsf{cov}} \to \mathcal{U} \tag{11}$$

is also covariant. Note that (11) can be restated as $C \equiv \lambda(B:\mathcal{U}_{\mathsf{cov}}).\,\mathsf{El}(A) \to \mathsf{El}(B)$. Finally, we appeal to theorem 2.7.1, the Yoneda lemma, to define homtofun:

$$\mathsf{homtofun} :\equiv \tag{12}$$

$$\mathsf{yon}^C_{\mathsf{El}(A)}(\mathsf{id}_{\mathsf{El}(A)})(\mathsf{El}(B)) :\equiv \tag{13}$$

$$\lambda f.\, f_*(\mathsf{id}_{\mathsf{El}(A)}) : \mathsf{hom}_{\mathcal{U}_{\mathsf{cov}}}\,(A,B) \to (\mathsf{El}(A) \to \mathsf{El}(B)) \tag{14}$$

□

We first observe that homtofun behaves like a functor, which is formalized by the following lemma:

Lemma 2.8.2. *For all $A, B, C : \mathcal{U}_{\mathsf{cov}}$ along with $f : \mathsf{hom}_{\mathcal{U}_{\mathsf{cov}}}\,(A,B)$ and $g : \mathsf{hom}_{\mathcal{U}_{\mathsf{cov}}}\,(B,C)$,*

$$\mathsf{homtofun}(g \cdot f) = \mathsf{homtofun} g \circ \mathsf{homtofun}\, f$$

Proof.

$$\begin{aligned}\mathsf{homtofun}(g \cdot f) : \mathsf{El}(A) \to \mathsf{El}(C) &\equiv \\ (g \cdot f)_*(\mathsf{id}_{\mathsf{El}(A)}) &= \\ g_*(f_*(\mathsf{id}_{\mathsf{El}(A)})) &\equiv \\ \lambda(x : \mathsf{El}(A)).\, (\mathsf{homtofun}(g))((\mathsf{homtofun}(f))(x)) &\equiv \\ \mathsf{homtofun}(g) \circ \mathsf{homtofun}(f) : \mathsf{El}(A) \to \mathsf{El}(C)\end{aligned}$$

with the second equality following from theorem 2.6.7. □

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 2.8.3 (Directed Univalence)**.** *For any points $A, B : \mathcal{U}_{\mathsf{cov}}$, the function defined in Lemma theorem 2.8.1 is an equivalence.*

For points $A, B : \mathcal{U}_{\mathsf{cov}}$, we can break the equivalence up into:

- An introduction rule for $\mathsf{hom}_{\mathcal{U}_{\mathsf{cov}}}(A, B)$ denoted dua for "directed univalence axiom":

$$\mathsf{dua} : (\mathsf{El}(A) \to \mathsf{El}(B)) \to (\mathsf{hom}_{\mathcal{U}_{\mathsf{cov}}}(A, B))$$

- An elimination rule

$$\mathsf{homtofun} :\equiv \mathsf{yon}^{C}_{\mathsf{El}(A)}(\mathsf{id}_{\mathsf{El}(A)}) : \mathsf{hom}_{\mathcal{U}_{\mathsf{cov}}}(A, B) \to (\mathsf{El}(A) \to \mathsf{El}(B))$$

- The propositional computation rule

$$\mathsf{homtofun}(\mathsf{dua}(f)) = f$$

- The propositional uniqueness rule, which states for any $f : \mathsf{hom}_{\mathcal{U}_{\mathsf{cov}}}(A, B)$,

$$f = \mathsf{dua}(\mathsf{homtofun}(f))$$

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

$$\begin{aligned}\mathsf{idhom}_A &= \mathsf{dua}(\mathsf{id}_{\mathsf{El}(A)}) \\ \mathsf{dua}g \cdot \mathsf{dua}\, f &= \mathsf{dua}(g \circ f).\end{aligned}$$

The first equality from the fact that

$$\begin{aligned}\mathsf{id}_{\mathsf{El}(A)} &= \\ (\mathsf{idhom}_A)_*(\mathsf{id}_A) &\equiv \\ \mathsf{homtofun}(\mathsf{idhom}_A)\end{aligned}$$

by theorem 2.6.7. To show the second, if we define $p :\equiv \mathsf{dua}(f)$ and $q :\equiv \mathsf{dua}(g)$, then

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

With directed univalence, we can show that dependent transport is really just coercing a morphism of types.

Lemma 2.8.4. *For any map $P : A \to \mathcal{U}$ and morphism $f : \mathsf{hom}_A(x, y)$,*

$$\mathsf{dua}(f_*) = P(f)$$

Proof. We have the following chain of equalities:

$$\begin{aligned} \mathsf{dua}(f_*) &\equiv \\ \mathsf{dua}(\mathsf{dtransport}^{\mathsf{El}(P)}(f, -)) &= \\ \mathsf{dua}(\mathsf{dtransport}^{\mathsf{El}}(P(f), -)) &\equiv \\ \mathsf{dua}(\mathsf{homtofun}(P(f))) &= \\ P(f) & \end{aligned}$$

with the first following from lemma theorem 2.6.5 which allows us to decompose the transport of a composition into an application and a coercion. The second equality comes from the propositional uniqueness rule. □

We can also show that a morphism between functions is a commutative square.

Lemma 2.8.5. *For any two maps $F, G : A \to \mathcal{U}_{\mathsf{cov}}$, a morphism $h : \mathsf{hom}_A(x, y)$ and maps $f : \mathsf{El}(F(x)) \to \mathsf{El}(G(x))$, and $g : \mathsf{El}(F(y)) \to \mathsf{El}(G(y))$ we have the following:*

$$\mathsf{hom}_{(\lambda x.\, \mathsf{El}(F(x)) \to \mathsf{El}(G(x)))h}(f, q) \simeq g \circ h_* = h_* \circ f$$

Proof. By directed univalence,

$$\mathsf{hom}_{(\lambda x.\, \mathsf{El}(F(x)) \to \mathsf{El}(G(x)))h}(f, q) \simeq \mathsf{hom}_{(\lambda x.\, \mathsf{hom}_{\mathcal{U}_{\mathsf{cov}}}(F(x), G(x)))h}(\mathsf{dua}\, f, \mathsf{dua}\, g)$$

By lemma theorem 2.3.15, this is equivalent to

$$\mathsf{dua}\, g \cdot F(h) = G(h) \cdot \mathsf{dua}\, f$$

which is equivalent to

$$\mathsf{dua}\, g \cdot \mathsf{dua} h_* = \mathsf{dua} h_* \cdot \mathsf{dua}\, f$$

by lemma theorem 2.8.4. The functorality of dua allow us to push this equivalence to the type

$$\mathsf{dua}(g \circ h_*) = \mathsf{dua}(h_* \circ f).$$

From there, we can apply $\mathsf{homtofun}$ to both sides, and noting that $\mathsf{homtofun}$ and dua are inverses, extending the equivalence to the type

$$g \circ h_* = h_* \circ f$$

by lemma theorem 1.4.4. □

Directed univalence also extends to characterizing dependent functions as morphisms in $\mathcal{U}_{\mathsf{cov}}$.

Lemma 2.8.6. *Given maps $P : A \to \mathcal{U}_{\mathsf{cov}}$ and $Q : \widetilde{\mathsf{El}(P)} \to \mathcal{U}_{\mathsf{cov}}$, for all $x : A$ there is an equivalence*

$$\prod_{p:\mathsf{El}(P(x))} \mathsf{El}(Q(x,p)) \simeq \sum_{f:\mathsf{hom}_{\mathcal{U}_{\mathsf{cov}}}\left(P(x),\widetilde{\sum_{(p:\mathsf{El}(P(x)))}\mathsf{El}(Q(x,p))}\right)} \mathsf{dua}(\mathsf{fst})\cdot f = \mathsf{idhom}_{P(x)},$$

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

Proof. We begin by noting the following equivalences

$$\sum_{f:\mathsf{hom}_{\mathcal{U}_{\mathsf{cov}}}\left(P(x),\widetilde{\sum_{(p:P(x))}Q(x,p)}\right)} \mathsf{dua}(\mathsf{fst}) \circ f = \mathsf{idhom}_{P(x)} \simeq$$

$$\sum_{f:\mathsf{El}(P(x))\to\sum_{(p:\mathsf{El}(P(x)))}\mathsf{El}(Q(x,p))} \mathsf{dua}(\mathsf{fst}) \circ \mathsf{dua}(f) = \mathsf{idhom}_{P(x)} \simeq$$

$$\sum_{f:\mathsf{El}(P(x))\to\sum_{(p:\mathsf{El}(P(x)))}\mathsf{El}(Q(x,p))} \mathsf{dua}(\mathsf{fst} \circ f) = \mathsf{idhom}_{P(x)} \simeq$$

$$\sum_{f:\mathsf{El}(P(x))\to\sum_{(p:\mathsf{El}(P(x)))}\mathsf{El}(Q(x,p))} \mathsf{fst} \circ f = \mathsf{id}_{\mathsf{El}(P(x))}$$

with the first equivalence being a change of base type to an equivalence type thus following from lemma theorem 1.4.7. The second equivalence is a result of the functorality of dua and lemmas theorem 1.4.8 and theorem 1.4.6. The third equivalence comes from theorem 1.4.4 as since $\mathsf{homtofun}$ is an equivalence, thus an embedding. Now that we are working with functions, we can reduce the types further:

$$\sum_{f:\mathsf{El}(P(x))\to\sum_{(p:\mathsf{El}(P(x)))}\mathsf{El}(Q(x,p))} \mathsf{fst} \circ f = \mathsf{id}_{\mathsf{El}(P(x))} \simeq$$

$$\sum_{(f_1,f_2):\sum_{(g:\mathsf{El}(P(x))\to\mathsf{El}(P(x)))}\prod_{(p:\mathsf{El}(P(x)))} El(Q(x,g(p)))} f_1 = \mathsf{id}_{\mathsf{El}(P(x))} \simeq$$

$$\sum_{f:\mathsf{El}(P(x))\to\mathsf{El}(P(x))} \left(\prod_{p:\mathsf{El}(P(x))} Q(x,f(p))\right) \times (f = \mathsf{id}_{\mathsf{El}(P(x))}) \simeq$$

$$\sum_{f:\mathsf{El}(P(x))\to\mathsf{El}(P(x))} (f = \mathsf{id}_{\mathsf{El}(P(x))}) \times \left(\prod_{p:\mathsf{El}(P(x))} \mathsf{El}(Q(x,f(p)))\right) \simeq$$

$$\sum_{((f,p):\sum_{(g:\mathsf{El}(P(x))\to\mathsf{El}(P(x)))} g=\mathsf{id}_{\mathsf{El}(P(x))})} \prod_{(p:\mathsf{El}(P(x)))} \mathsf{El}(Q(x,f(p))) \simeq$$

$$\prod_{p:\mathsf{El}(P(x))} \mathsf{El}(Q(x,p))$$

with the first equivalence following from the type theoretic axiom of choice, lemma theorem 1.5.9. The second is an application of associativity, thus follows

from lemma theorem 1.8.2. The third follows from the symmetry of the non-dependent pair type, lemma theorem 1.8.1. The fourth is another application of associativity thus follows again from lemma theorem 1.8.2. Note that the base is now a contractible singleton, thus contractible by lemma theorem 1.5.7, and we can simplify the sigma type with a contractible base by lemma theorem 1.5.8. Thus, we can get our desired equivalence by composing the two equivalences formed in this proof. □

2.9 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* to a 2-simplex in the total space. Before we give the condition for a family to be inner, we must make sense of what it means to be a "2-simplex in the total space".

Definition 2.9.1. Given a type A and

$$\begin{gathered} x, y, z : A \\ f : \mathsf{hom}_A(x, y) \qquad g : \mathsf{hom}_A(y, z) \qquad h : \mathsf{hom}_A(x, z) \\ t : \hom_A^2 \left(\begin{array}{ccccc} & & y & & \\ & f \nearrow & & \searrow g & \\ x & & \xrightarrow[h]{} & & z \end{array} \right) \\ u : C(x) \qquad v : C(y) \qquad w : C(z) \\ p : \hom_{C(f)}(u, v) \qquad q : \hom_{C(g)}(v, w) \qquad r : \hom_{C(h)}(u, w) \end{gathered}$$

we define

$$\hom^2_{C(t)} \left(\begin{array}{ccccc} & & v & & \\ & p \nearrow & & \searrow q & \\ u & & \xrightarrow[r]{} & & w \end{array} \right) :\equiv \left\langle \textstyle\prod_{s:\Delta^2} C(t(s)) \Big|^{\partial\Delta^2}_{[u,v,w,p,q,r]} \right\rangle$$

and say that a morphism in $\hom^2_{C(t)} \left(\begin{array}{ccccc} & & v & & \\ & p \nearrow & & \searrow q & \\ u & & \xrightarrow[r]{} & & w \end{array} \right)$ is a **dependent 2-simplex over** t.

With that, we can precisely state what it means for a type family to be inner.

Definition 2.9.2. For any type A, type family $C : A \to \mathcal{U}$, and

$$\begin{gathered} x, y, z : A \\ f : \mathsf{hom}_A(x, y) \qquad g : \mathsf{hom}_A(y, z) \qquad h : \mathsf{hom}_A(x, z) \\ t : \hom_A^2 \left(\begin{array}{ccccc} & & y & & \\ & f \nearrow & & \searrow g & \\ x & & \xrightarrow[h]{} & & z \end{array} \right) \\ u : C(x) \qquad v : C(y) \qquad w : C(z) \\ p : \hom_{C(f)}(u, v) \qquad q : \hom_{C(g)}(v, w) \end{gathered}$$

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

$$\sum_{r:\hom_{C(h)}(u,w)} \hom^2_{C(t)} \left(\begin{array}{ccccc} & & v & & \\ & p \nearrow & & \searrow q & \\ u & & \xrightarrow[r]{} & & w \end{array} \right) \tag{15}$$

is contractible.

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

Lemma 2.9.3 ([BW22], 4.1.5)**.** *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 2.9.4 ([BW22], 6.1.1)**.** *Every covariant type family is an inner type family.*

Lemma 2.9.5. *Given a Segal type A, $P : A \to \mathcal{U}_{\mathsf{cov}}$, and $Q : \widetilde{\mathsf{El}(P)} \to \mathcal{U}_{\mathsf{cov}}$ the type family*

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

is inner.

Proof. Since A is Segal, by lemma theorem 2.9.3 it suffices to show

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

is Segal. Note that

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

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

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

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

$$\left\langle \textstyle\prod_{t:\Delta^2} \prod_{(p:\mathsf{El}(P(\mathsf{comp}_{g,f}(t))))} \mathsf{El}(Q(\mathsf{comp}_{g,f}(t), p)) \middle|^{\Lambda^2_1}_{\psi} \right\rangle \tag{16}$$

where g, f make up the legs of ϕ. We seek to show that this type is contractible. By lemma theorem 2.8.6, shortening definitions by taking

$$\zeta :\equiv \mathsf{comp}_{g,f}$$

$$\epsilon :\equiv \mathsf{hom}_{\mathcal{U}_{\mathsf{cov}}}\left(P(\zeta(t)), \overline{\sum_{p:\mathsf{El}(P(\zeta(t)))} \mathsf{El}(Q(\zeta(t),p))}\right)$$

$$\mathsf{fst} : \left(\sum_{p:\mathsf{El}(P(\zeta(t)))} \mathsf{El}(Q(\zeta(t),p))\right) \to \mathsf{El}(P(\zeta(t))),$$

this type is equivalent to

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

By lemma theorem 2.3.14, this is equivalent to the extension type

$$\left\langle \prod_{t:\Delta^2} \sum_{(f:\epsilon)} \hom^2_{\mathcal{U}_{\mathsf{cov}}}(\mathsf{dua}(\mathsf{fst}), f; \mathsf{idhom}_{P(\zeta(t))}) \middle| \begin{matrix} \Lambda^2_1 \\ \psi'' \end{matrix} \right\rangle$$

Let $\alpha \equiv \lambda t.\, [(P(\zeta(t)), \overline{\sum_{(p:P(\zeta(t)))} Q(\zeta(t),p)}, P(\zeta(t)), \mathsf{dua}(\mathsf{fst}), \mathsf{idhom}_{P(\zeta(t))}]$. We can push our equivalence to the type

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

by lemma eq. (6) as we are taking the sum over a side of a 2-morphism. We then proceed by applying lemma theorem 2.2.2 which allows us to swap the order of the nested extension types:

$$\left\langle \prod_{s:\Delta^2} \left\langle \prod_{t:\Delta^2} \mathcal{U}_{\mathsf{cov}} \middle| \begin{matrix} \Lambda^2_1 \\ \lambda t.\, \psi'''(t)(s) \end{matrix} \right\rangle \middle| \begin{matrix} \Lambda^2_2 \\ \lambda t.\, \alpha(t) \end{matrix} \right\rangle \tag{17}$$

Since $\mathcal{U}_{\mathsf{cov}}$ is Segal, the fibers of eq. (17) are contractible. Thus, the entire type in eq. (17) is contractible by relative functional extensionality. So, eq. (16) is contractible as well, since it is equivalent to eq. (17). □

3 Directed Higher Inductive Types

In this section, we start our investigation of directed higher inductive types, starting at types that have no simplicial data to types that have more complicated structure. For these types, our main object of study is the generalization of the fundamental groupoid of a space: the **fundamental monoid** of a category.

3.1 Inductive types in sHoTT

define set, characterize path space of nd, show it is a set.

As a warmup to more complicated directed higher inductive types, in this section we'll study the degenerate case of a dHIT: inductive types. More specifically, we study the natural numbers. A naive statement of the naturals in sHOTT is as follows:

Definition 3.1.1. Let $\mathbb{N}$ be the type generated by the constructors

- $0 : \mathbb{N}$
- For all $n : \mathbb{N}$, a term $\mathsf{succ}(n) : \mathbb{N}$.

The induction principle for $\mathbb{N}$ states that for any type family $C : \mathbb{N} \to \mathcal{U}$ along with

- A point $c_0 : C(0)$
- A function $c_s : \prod_{(n:\mathbb{N})} C(n) \to C(\mathsf{succ}(n))$

There is a function $f : \prod_{(n:\mathbb{N})} C(n)$ such that $f(0) \equiv c_0$ and for all $n : \mathbb{N}$, $f(\mathsf{succ}(n)) \equiv c_s(n, f(n))$.

While this incarnation of the natural numbers seems syntactically permissible, its behavior is poor. For example, we can try to characterize the morphism type between two arbitrary natural numbers. To do so, we define a map $\mathsf{code} : \mathbb{N} \to \mathbb{N} \to \mathcal{U}$ with the defining equations

$$\begin{aligned}
&\mathsf{code}\,0\,0 :\equiv \mathsf{Unit}\\
&\mathsf{code}\,0\,S(n) :\equiv \mathsf{Void}\\
&\mathsf{code}\,S(n)\,0 :\equiv \mathsf{Void}\\
&\mathsf{code}\,S(n)\,S(m) :\equiv \mathsf{code}\,n\,m
\end{aligned}$$

We immediately run into issues while trying to define

$$\mathsf{encode} : \prod_{(n:\mathbb{N})} \prod_{(m:\mathbb{N})} \mathsf{hom}_{\mathbb{N}}(n, m) \to \mathsf{code}\,n\,m.$$

To do so, we'd have to supply a term for $\mathsf{encode}\,0\,S(n)$ and $\mathsf{encode}\,S(n)\,0$. Evaluating the type of such a term, we find that we are required to come up with a map that takes a morphism of type $\mathsf{hom}_{\mathbb{N}}(0, S(n))$ to a term of type $\mathsf{code}\,0\,S(n) \equiv \mathsf{Void}$. Since we cannot inhabit Void, we need to show that $\mathsf{hom}_{\mathbb{N}}(0, S(n))$ is also empty. That is, we would need to be able to characterizations the morphisms of $\mathbb{N}$ to define encode. Such a requirement is circular, thus untenable.

The issue lies in our naive implementation of the natural numbers. Since we have positively described $\mathbb{N}$ with no simplicial or homotopical data (morphisms or paths), it is reasonable to think that the type *should* be Segal or discrete. Of course, if we could characterize the morphisms in the naturals then a proof of

Segal or discrete structure would be straightforward. As we'll see in the next section, all directed higher inductive types have the structure of a Segal type. Thus, a workaround for us is to include the Segal constructur to the natural numbers definition and observe the consequences.

Definition 3.1.2. Let $\mathbb{N}^s$ be the *Segal* type generated by the constructors

- $0 : \mathbb{N}^s$
- For all $n : \mathbb{N}^s$, a term $\mathsf{succ}(n) : \mathbb{N}^s$.

The induction principle for $\mathbb{N}^s$ states that for any inner type family $C : \mathbb{N}^s \to \mathcal{U}$ along with

- A point $c_0 : C(0)$
- A function $c_s : \prod_{(n:\mathbb{N}^s)} C(n) \to C(\mathsf{succ}(n))$

There is a function $f : \prod_{(n:\mathbb{N}^s)} C(n)$ such that $f(0) \equiv c_0$ and for all $n : \mathbb{N}^s$, $f(\mathsf{succ}(n)) \equiv c_s(n, f(n))$.

The elimination principles ensure that we can only eliminate into Segal types. We also define a natural numbers object that carries the discrete constructor, with elimination rules ensuring that it can only eliminate into discrete types:

Definition 3.1.3. Let $\mathbb{N}^d$ be the *discrete* type generated by the constructors

- $0 : \mathbb{N}^d$
- For all $n : \mathbb{N}^d$, a term $\mathsf{succ}(n) : \mathbb{N}^d$.

The induction principle for $\mathbb{N}^d$ states that for any point-wise discrete type family $C : \mathbb{N}^d \to \mathcal{U}$ along with

- A point $c_0 : C(0)$
- A function $c_s : \prod_{(n:\mathbb{N}^d)} C(n) \to C(\mathsf{succ}(n))$

There is a function $f : \prod_{(n:\mathbb{N}^d)} C(n)$ such that $f(0) \equiv c_0$ and for all $n : \mathbb{N}^d$, $f(\mathsf{succ}(n)) \equiv c_s(n, f(n))$.

We observe that $\mathbb{N}^d$ is Segal because discrete spaces are Segal (lemma theorem 2.4.2). On the other hand, we can show that $\mathbb{N}^s$ is discrete:

Lemma 3.1.4. *$\mathbb{N}^s$ is discrete.*

Proof. Given that $\mathbb{N}^s$ is Segal, we can characterize the morphism space in a way that looks similar to book HoTT characterization of the path space of $\mathbb{N}$. Define a map

$$\mathsf{code} : \mathbb{N}^s \to \mathbb{N}^s \to \mathcal{U}_{\mathsf{cov}}$$

as

$$\begin{aligned}
&\mathsf{code}\,0\,0 :\equiv \overline{\mathsf{Unit}}\\
&\mathsf{code}\,0\,S(n) :\equiv \overline{\mathsf{Void}}\\
&\mathsf{code}\,S(n)\,0 :\equiv \overline{\mathsf{Void}}\\
&\mathsf{code}\,S(n)\,S(m) :\equiv \mathsf{code}\,n\,m
\end{aligned}$$

There is a dependent function $r : \prod_{(n:\mathbb{N}^s)} \mathsf{El}(\mathsf{code}\,n\,n)$ defined as

$$\begin{aligned}
r\,0 &:\equiv \star\\
r\,S(n) &:\equiv r(n)
\end{aligned}$$

We can define $\mathsf{encode} : \prod_{(n:\mathbb{N}^s)} \prod_{(m:\mathbb{N}^s)} \mathsf{hom}_{\mathbb{N}^s}\,(n,m) \to \mathsf{El}(\mathsf{code}\,n\,m)$ as

$$\mathsf{encode}\,n :\equiv \mathrm{yon}_n^{\lambda m.\,\mathsf{El}(\mathsf{code}\,n\,m)}(r(n))$$

and $\mathsf{decode} : \prod_{(n:\mathbb{N}^s)} \prod_{(m:\mathbb{N}^s)} \mathsf{El}(\mathsf{code}\,n\,m) \to \mathsf{hom}_{\mathbb{N}^s}\,(n,m)$ as

$$\begin{aligned}
&\mathsf{decode}\,0\,0\,_ :\equiv \mathsf{idhom}_0\\
&\mathsf{decode}\,S(n)\,S(m)\,u :\equiv \lambda i.\,S((\mathsf{decode}\,n\,m\,u)i)
\end{aligned}$$

We must now prove that encode and decode are quasi-inverses. We do so by defining a map

$$\eta : \prod_{(n:\mathbb{N}^s)} \prod_{(m:\mathbb{N}^s)} \prod_{(f:\mathsf{hom}_{\mathbb{N}^s}(n,m))} \mathsf{decode}\,n\,m\,(\mathsf{encode}\,n\,m\,f) = f$$

as

$$\eta\,n :\equiv \mathrm{yon}_n^{\lambda(m,f).\,\mathsf{decode}\,n\,m\,(\mathsf{encode}\,n\,m\,f)=f}(\mathsf{refl}_{\mathsf{idhom}_0}) \tag{1}$$

with this being a valid definition as $\lambda(m,f).\,\mathsf{decode}\,n\,m\,(\mathsf{encode}\,n\,m\,f) = f$ is covariant due to the covariance of $\lambda m.\,\mathsf{hom}_{\mathbb{N}^s}\,(n,m)$, which follows from the assumption that $\mathbb{N}^s$ is Segal.

To define the final term showing that encode and decode are quasi-inverses, we define a map showing that the fibers of El(code) are mere propositions. To do so, we define the following map $\psi : \prod_{(n,m:\mathbb{N}^s)} \prod_{(u,v:\mathsf{El}(\mathsf{code}\,n\,m))} u =_{\mathsf{El}(\mathsf{code}\,n\,m)} v$ as

$$\begin{aligned}
&\psi\,0\,0\,\star\,\star :\equiv \mathsf{refl}_\star\\
&\psi\,S(n)\,S(m)\,u\,v :\equiv \psi(n\,m\,u\,v)
\end{aligned}$$

and define

$$\epsilon : \prod_{(n:\mathbb{N}^s)} \prod_{(m:\mathbb{N}^s)} \prod_{(u:\mathsf{El}(\mathsf{code}\,n\,m))} \mathsf{encode}\,n\,m\,(\mathsf{decode}\,n\,m\,u) = u$$

as

$$\epsilon\,n\,m\,u :\equiv \psi(n\,m\,(\mathsf{encode}\,n\,m\,(\mathsf{decode}\,n\,m\,u))\,u)$$

Thus, for all $n, m : \mathbb{N}^s$ it is the case that

$$\mathsf{hom}_{\mathbb{N}^s}(n, m) \simeq \mathsf{El}(\mathsf{code}\, n\, m).$$

We can similarly use $\mathsf{El}(\mathsf{code})$ to characterize the path space of $\mathbb{N}^s$ as well. Since this is a routine calculation that follows very similarly to the one we see above, we will only highlight the differences. The maps encode and η will follow similarly, except they will use transport as opposed to the Yoneda lemma. On the other hand, decode and ϵ will follow identitally. From that description, we can further conclude that

$$m =_{\mathbb{N}^s} n \simeq \mathsf{El}(\mathsf{code}\, n\, m) \simeq \mathsf{hom}_{\mathbb{N}^s}(n, m)\,.$$

So, $\mathbb{N}^s$ is discrete. □

Since it is clear that $\mathbb{N}^d$ is Segal, and we have shown that $\mathbb{N}^s$ is discrete, we can craft maps $f : \mathbb{N}^d \to \mathbb{N}^s$ and $g : \mathbb{N}^s \to \mathbb{N}^d$ by induction. Both maps will essentially be the identity function. Thus, it is clear that $\mathbb{N}^s$ and $\mathbb{N}^d$ are equivalent. Since $\mathbb{N}^d$ has the more restrictive elimination principle, we opt to use this as our canonical natural numbers object.

3.2 Directed Interval

For our first example of a directed higher inductive type, we start with the *directed interval*.

Let I be the *Segal* type generated by constructors

- $0_I : I$
- $1_I : I$
- $\mathsf{seg} : \mathsf{hom}_I(0_I, 1_I)$.

One thing to highlight of this definition, which is a common feature of all our directed higher inductive types, is the assumption of Segalness. In HoTT, no such assumption was needed. This harps on a theme threaded throughout this thesis. In HoTT, all types are groupoids, while in sHoTT, only some types are categories. Thus, if we wish to talk about free categories generated by constructors, it is not enough to just state the constructors; in such a type there is no guarantee that we can compose morphisms. We must work in a setting where we have a composition map, which the Segal constructors enforce. With this digression over, we can discuss the elimination principles of the type.
The recursion principle for I states that for any Segal type B with

- a point b_0
- a point b_1
- a morphism $s : \mathsf{hom}_B(b_0, b_1)$,

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

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 $s : \mathsf{hom}_{C(seg)}\,(c_0, c_1)$,

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 $f(\mathsf{seg}) = s$.

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

Lemma 3.2.1. *If B is a Segal type with $b_0 : B$, $b_1 : B$ and $s : \mathsf{hom}_B\,(b_0, b_1)$, then there is a function $f : I \to B$ satisfying*

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

Proof. We first define the type family $C :\equiv \lambda\_.\, B : I \to \mathcal{U}$. C is trivially an inner type family. So, we can use the induction principle, noting

$$b_0 : C(0_I) \equiv B$$

$$b_1 : C(1_I) \equiv B$$

$$s : \mathsf{hom}_{C(\mathsf{seg})}\,(b_0, b_1) \equiv \mathsf{hom}_B\,(b_0, b_1)$$

thus there is a function

$$f : \prod_{x:I} C(x) \equiv I \to B$$

satisfying our desired equations. □

Moreover, we can also prove a uniqueness principle:

Lemma 3.2.2. *If B is a Segal type and $f, g : I \to B$ are two maps along with equalities*

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

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

Proof. Let $C :\equiv \lambda x.\, f(x) =_B g(x)$. Observe that

$$p : f(0_I) =_B g(0_I) \equiv C(0_I)$$

$$q : f(1_I) =_B g(1_I) \equiv C(1_I)$$

and

$$r : \mathsf{hom}_{(\lambda x.\, f(x)=g(x))\mathsf{seg}}\,(p, q) \equiv \mathsf{hom}_{C(\mathsf{seg})}\,(p, q)\,.$$

Thus, we can define a function $\alpha : \prod_{(x:I)} C(x) \equiv \prod_{(x:I)} f(x) = g(x)$ as desired. □

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 $\mathcal{U}_{\mathsf{cov}}$ is Segal, we use the recursion principle for I to define a type family $\mathsf{code} : I \to \mathcal{U}_{\mathsf{cov}}$ with the defining equations:

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

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

Lemma 3.2.3. $\mathsf{hom}_I\,(1_I, 0_I) \simeq \mathsf{Void}$

Proof. For any $f : \mathsf{hom}_I\,(1_I, 0_I)$, we note that $\mathsf{dtransport}^{\mathsf{El}(\mathsf{code})}(f, \star) : \mathsf{Void}$, which is enough to say that $\mathsf{hom}_I\,(1_I, 0_I) \simeq \mathsf{Void}$ □

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

Lemma 3.2.4. *There is a function*

$$\mathsf{encode} : \prod_{(x:I)} \mathsf{hom}_I\,(1_I, x) \to \mathsf{El}(\mathsf{code}(x))$$

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

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

□

Since I is Segal, code and $\lambda x.\overline{\mathsf{hom}_I\,(1_I, x)}$ both have type $I \to \mathcal{U}_{\mathsf{cov}}$, thus the type family $\lambda x.\mathsf{El}(\mathsf{code}(x)) \to \mathsf{hom}_I\,(1_I, x)$ is inner. With that, and lemma theorem 2.9.5, we can define decode.

Lemma 3.2.5. *There is a function*

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

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

$$f :\equiv \lambda\, u.\mathrm{abort}(u)$$
$$g :\equiv \lambda\_.\mathsf{idhom}_{1_I}$$

which allows us to define the point constructors of decode as

$$\mathsf{decode}(0_I) = f$$
$$\mathsf{decode}(1_I) = g$$

By lemma theorem 2.8.5, to define $\mathsf{decode}(\mathsf{seg})$, it suffices to show that $\mathsf{seg}_* \circ f = g \circ \mathsf{seg}_*$. It follows from the fact that both sides explode. □

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

Lemma 3.2.6. *There is a map*

$$\eta : \Pi_{x:I}\, \Pi_{f:\mathsf{hom}_I\,(1_I,x)} \mathsf{decode}\, x\, (\mathsf{encode}\, x\, f) = f$$

Proof. Noting that $\lambda(x, f).\mathsf{decode}\, x\, (\mathsf{encode}\, x\, f) = f$ is covariant by corollary 2.6.13.1. We further observe that by theorem 2.7.2, it suffices to provide a term of the type

$$\mathsf{decode}\, 1_I\, (\mathsf{encode}\, 1_I\, \mathsf{idhom}_{1_I}) = \mathsf{idhom}_{1_I}\,.$$

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

$$\mathsf{decode}\, 1_I\, \star = \mathsf{idhom}_{1_I}\,.$$

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

$$\mathsf{idhom}_{1_I} = \mathsf{idhom}_{1_I}$$

of which $\mathsf{refl}_{\mathsf{idhom}_{1_I}}$ suffices. □

Lemma 3.2.7. *There is a map*

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

Proof. First, we note that $\mathsf{code} : I \to \mathcal{U}_{\mathsf{cov}}$ and $\mathcal{U}_{\mathsf{cov}}$ is closed under paths, so there is a map $\lambda((x, u) : \widetilde{\mathsf{El}(\mathsf{code})}).\, \overline{\mathsf{encode}\, x\, (\mathsf{decode}\, x u) = u} : I \to \mathcal{U}_{\mathsf{cov}}$. Thus, the type family $\lambda x.\, \Pi_{u:\mathsf{El}(\mathsf{code}(x))} \mathsf{encode}\, x\, (\mathsf{decode}\, x u) = u$ is inner by theorem 2.9.5. Proceeding with the proof, we partially define ϵ as follows:

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

To complete the definition, we must define a term

$$\epsilon(\mathsf{seg}) : \mathsf{hom}_{(\lambda x.\Pi_{u:\mathsf{El}(\mathsf{code}(x))}\mathsf{encode}\, x\,(\mathsf{decode}\, x\, u)=u)\mathsf{seg}}\,(f, g)\,.$$

By lemma theorem 2.6.6, to inhabit the dependent morphism over seg, we can provide a path in the type

$$\mathsf{seg}_*(f) =_{\Pi_{u:\mathsf{El}(\mathsf{code}(1_I))}\mathsf{encode}\, 1_I\,(\mathsf{decode}\, 1_I\, u)=u}\; g$$

which is definitionally equal to the type

$$\mathsf{seg}_*(f) =_{\Pi_{u:\mathsf{Unit}}\star=u}\; g.$$

By lemma theorem 1.7.1, function extensionality, it suffices to define a term of type

$$\prod_{u:\mathsf{Unit}} \mathsf{seg}_*(f)(u) =_{\star=u}\; g(u).$$

To define the map, assume $u : \mathsf{Unit}$. By lemma theorem 1.5.6, the contractability of Unit passes onto its identity types. So, $\star =_{\mathsf{Unit}} u$ is also contractible. Thus, there is an inhabitant of the type

$$seg_*(f)(u) =_{\star=u}\; g(u).$$

We can name this function $\phi : \prod_{(u:\mathsf{Unit})} \mathsf{seg}_*(f)(u) =_{\star=u}\; g(u)$ and trace it back the chain of equivalences to define a morphism

$$\phi' : \mathsf{hom}_{(\lambda x.\Pi_{u:\mathsf{El}(\mathsf{code}(x))}\mathsf{encode}\, x\,(\mathsf{decode}\, x\, u)=u)\mathsf{seg}}\,(f, g)\,.$$

We can then complete the definition of ϵ with

$$\epsilon(\mathsf{seg}) = \phi'$$

. □

3.3 Directed Circle

We can also define the **direted circle**, similarly known as the **simplicial circle** and show that its fundamental monoid is the natural numbers.

Let S be the *Segal* type generated by the constructors

- $\mathsf{base} : S$
- $\mathsf{dloop} : \mathsf{hom}_S\,(\mathsf{base}, \mathsf{base})$

The recursion principle for S states that for any Segal type B along with

- a point b : B
- a morphism $l : \mathsf{hom}_{B}\,(b, b)$

there is a function $f : S \to B$ such that $f(\mathsf{base}) \equiv b$ and $f(\mathsf{dloop}) = l$.

The induction principle for S states that for any inner type family $C : S \to \mathcal{U}$ along with

- a point $c : C(\mathsf{base})$
- a dependent morphism $l : \mathsf{hom}_{C(\mathsf{dloop})}\,(c, c)$

there is a function $f : \prod_{(x:S)} C(x)$ such that $f(\mathsf{base}) \equiv c$ and $f(\mathsf{dloop}) = l$.

Just like before, we start by defining a function $\mathsf{code} : S \to \mathcal{U}_{\mathsf{cov}}$ with the defining equations

$$\begin{aligned}\mathsf{code}\,\mathsf{base} &:\equiv \overline{\mathbb{N}} \\ \mathsf{code}\,\mathsf{dloop} &:\equiv \mathsf{dua}(\mathsf{succ})\end{aligned}$$

From there, we can define the first part of our encode-decode pattern:

Lemma 3.3.1. *There is a function*

$$\mathsf{encode} : \prod_{x:S} \mathsf{hom}_{S}\,(\mathsf{base}, x) \to \mathsf{El}(\mathsf{code}(x))$$

Proof. We can define encode as

$$\mathsf{encode} = \mathsf{yon}^{\mathsf{El}(\mathsf{code})}_{\mathsf{base}}(0)$$

□

Then, we define the second part of our encode-decode pattern:

Lemma 3.3.2. *There is a map*

$$\mathsf{decode} : \prod_{x:S} \mathsf{El}(\mathsf{code}(x)) \to \mathsf{hom}_{S}\,(\mathsf{base}, x)$$

Proof. Before we can define decode, we define a map $f : \mathbb{N}^d \to \mathsf{hom}_{S}\,(\mathsf{base}, \mathsf{base})$ as

$$\begin{aligned}f(0) &:\equiv \mathsf{idhom}_{\mathsf{base}} \\ f(\mathsf{succ}(n)) &:\equiv \mathsf{dloop} \cdot f(n)\end{aligned}$$

This allows us to define the point constructor of decode with the following definition

$$\mathsf{decode}\,\mathsf{base} :\equiv f$$

To define decode(dloop), we need to supply a term of type

$$\mathsf{dua}(f) \cdot \mathsf{dua}(\mathsf{succ}) = \mathsf{hom}_{s}\,(\mathsf{base}, \mathsf{dloop}) \cdot \mathsf{dua}(f),$$

which is equivalent to the type

$$f \circ \mathsf{succ} = \mathsf{dloop}_* \circ f$$

For all $n : \mathbb{N}$, we have that $(f \circ \mathsf{succ})(n) \equiv \mathsf{dloop} \cdot f(n)$ and $(\mathsf{dloop}_* \circ f)(n) \equiv \mathsf{dloop} \cdot f(n)$. Thus, it suffices to supply refl.

□

With encode and decode defined, it is time to show they form an equivalence.

Lemma 3.3.3. *There is a map*

$$\eta : \prod_{(x:S)} \prod_{(f:\mathsf{hom}_S(\mathsf{base},x))} \mathsf{decode}\, x\, (\mathsf{encode}\, x\, f) = f$$

Proof. By the dependent yoneda lemma, it suffices to provide a term of the type

$$\mathsf{decode}\, \mathsf{base} (\mathsf{encode}\, \mathsf{base}\, \mathsf{idhom}_{\mathsf{base}}) = \mathsf{idhom}_{\mathsf{base}}.$$

Note, though, that encode was defined using the Yoneda lemma, hence it is a natural transformation that sends $\mathsf{idhom}_{\mathsf{base}}$ to 0. So, our equality becomes

$$\mathsf{decode}\, \mathsf{base}\, 0 = \mathsf{idhom}_{\mathsf{base}}.$$

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

$$\mathsf{idhom}_{\mathsf{base}} = \mathsf{idhom}_{\mathsf{base}}$$

of which $\mathsf{refl}_{\mathsf{idhom}_{\mathsf{base}}}$ suffices. □

Lemma 3.3.4. *There is a map*

$$\epsilon : \prod_{(x:S)} \prod_{(u:\mathsf{code}\, x)} \mathsf{encode}\, x\, (\mathsf{decode}\, x\, u) = u$$

Proof. We can leverage the induction principle to define this map. So, in the context of $x : S$, it suffices to first consider $x \equiv \mathsf{base}$. In that case, we can use natural number induction and it suffices to consider $u \equiv 0$ first. In that case, we note that

$$\mathsf{encode}\, \mathsf{base}\, (\mathsf{decode}\, \mathsf{base} 0) = 0 \equiv 0 = 0$$

So, we simply supply refl. When $u \equiv \mathsf{succ}(n)$, we note the following chain of paths:

$$\begin{aligned}
\mathsf{encode}\, \mathsf{base} (\mathsf{decode}\, \mathsf{base}\, \mathsf{succ}(n)) &\equiv \\
\mathsf{encode}\, \mathsf{base} (\mathsf{dloop} \cdot (\mathsf{decode}\, \mathsf{base}\, n)) &\equiv \\
\mathsf{encode}\, \mathsf{base} (\mathsf{dloop}_* (\mathsf{decode}\, \mathsf{base}\, n)) &= \\
\mathsf{dloop}_* (\mathsf{encode}\, \mathsf{base} (\mathsf{decode}\, \mathsf{base}\, n)) &= \\
\mathsf{dloop}_* (n) &= \\
\mathsf{succ}(n)
\end{aligned}$$

The concatenation inhabits $\mathsf{encode}\,\mathsf{base}(\mathsf{decode}\,\mathsf{base}\,\mathsf{succ}(n)) = \mathsf{succ}(n)$, as desired.

Now all is left is to show our lemma for $x \equiv \mathsf{dloop}$. To define the map, we must supply a term of type

$$\mathsf{hom}_{(\lambda x.\ \prod_{(u:\mathsf{code}(x))} \mathsf{encode}\,x(\mathsf{decode}\,x\,u)=u)(\mathsf{dloop})}\ (\mathsf{encode}(\mathsf{base}), \mathsf{encode}(\mathsf{base}))$$

which is equivalent to inhabiting the type

$$\mathsf{dloop}_*(\mathsf{encode}(\mathsf{base})) =_{\prod_{(u:\mathbb{N}^d)} \mathsf{encode}\,\mathsf{base}(\mathsf{decode}\,\mathsf{base}\,u)} \mathsf{encode}(\mathsf{base})$$

which by functional extensionality is equivalent to inhabiting the type

$$\prod_{u:\mathbb{N}^d} \mathsf{dloop}_*(\mathsf{encode}(\mathsf{base}))(u) = (\mathsf{encode}(\mathsf{base}))(u)$$

We can define such a function by noting that for all $u : \mathbb{N}^d$, the type

$$\mathsf{dloop}_*(\mathsf{encode}(\mathsf{base}))(u) =_{\mathsf{encode}\,\mathsf{base}\,(\mathsf{decode}\,\mathsf{base}\,u)=_{\mathbb{N}^d} u}\ (\mathsf{encode}(\mathsf{base}))(u)$$

is contractible, since $\mathbb{N}^d$ is a set. □

Thus, we have shown that the fundamental monoid of the simplicial circle is $\mathbb{N}^d$.

Appendix

$$\frac{}{\mathbf{1}\mathsf{cube}} \qquad \frac{I\,\mathsf{cube} \qquad J\,\mathsf{cube}}{I \times J\,\mathsf{cube}} \qquad \frac{(t : I) \in \Xi}{\Xi \vdash t : I} \qquad \frac{}{\Xi \vdash \star : \mathbf{1}}$$

$$\frac{\Xi \vdash s : I \qquad \Xi \vdash t : J}{\Xi \vdash \langle s, t \rangle : I \times J} \qquad \frac{\Xi \vdash t : I \times J}{\Xi \vdash \pi_1(t) : I} \qquad \frac{\Xi \vdash t : I \times J}{\Xi \vdash \pi_2(t) : J}$$

Figure 3: The cube layer

$$\frac{\phi \in \Phi}{\Xi \mid \Phi \vdash \phi} \qquad \frac{}{\Xi \vdash \top\mathsf{tope}} \qquad \frac{}{\Xi \mid \Phi \vdash \top} \qquad \frac{}{\Xi \vdash \bot\mathsf{tope}} \qquad \frac{\Xi \mid \Phi \vdash \bot}{\Xi \mid \Phi \vdash \psi}$$

$$\frac{\Xi \vdash \phi\mathsf{tope} \qquad \Xi \vdash \psi\mathsf{tope}}{\Xi \vdash (\phi \wedge \psi)\mathsf{tope}} \qquad \frac{\Xi \mid \Phi \vdash \phi \qquad \Xi \mid \Phi \vdash \psi}{\Xi \mid \Phi \vdash \phi \wedge \psi} \qquad \frac{\Xi \mid \Phi \vdash \phi \wedge \psi}{\Xi \mid \Phi \vdash \phi}$$

$$\frac{\Xi \mid \Phi \vdash \phi \wedge \psi}{\Xi \mid \Phi \vdash \psi} \qquad \frac{\Xi \vdash \phi\mathsf{tope} \qquad \Xi \vdash \psi\mathsf{tope}}{\Xi \vdash (\phi \vee \psi)\mathsf{tope}} \qquad \frac{\Xi \mid \Phi \vdash \phi}{\Xi \mid \Phi \vdash \phi \vee \psi}$$

$$\frac{\Xi \mid \Phi \vdash \psi}{\Xi \mid \Phi \vdash \phi \vee \psi} \qquad \frac{\Xi \mid \Phi, \phi \vdash \chi \qquad \Xi \mid \Phi, \psi \vdash \chi \qquad \Xi \mid \Phi \vdash \phi \vee \psi}{\Xi \mid \Phi \vdash \chi}$$

$$\frac{\Xi \vdash s : I \qquad \Xi \vdash t : I}{\Xi \vdash (s \equiv t)\mathsf{tope}} \qquad \frac{\Xi \vdash s : I}{\Xi \mid \Phi \vdash (s \equiv s)} \qquad \frac{\Xi \mid \Phi \vdash (s \equiv t)}{\Xi \mid \Phi \vdash (t \equiv s)}$$

$$\frac{\Xi \mid \Phi \vdash (s \equiv t) \qquad \Xi \mid \Phi \vdash (t \equiv v)}{\Xi \mid \Phi \vdash (s \equiv v)}$$

$$\frac{\Xi \mid \Phi \vdash (s \equiv t) \qquad \Xi, x : I \vdash \psi\mathsf{tope} \qquad \Xi \mid \Phi \vdash \psi[s/x]}{\Xi \mid \Phi \vdash \psi[t/x]} \qquad \frac{\Xi \vdash t : \mathbf{1}}{\Xi \mid \Phi \vdash t \equiv \star}$$

$$\frac{\Xi \vdash s : I \qquad \Xi \vdash t : J}{\Xi \mid \Phi \vdash \pi_1(\langle s, t\rangle) \equiv s} \qquad \frac{\Xi \vdash s : I \qquad \Xi \vdash t : J}{\Xi \mid \Phi \vdash \pi_2(\langle s, t\rangle) \equiv t}$$

$$\frac{\Xi \vdash t : I \times J}{\Xi \mid \Phi \vdash t \equiv \langle \pi_1(t), \pi_2(t)\rangle}$$

Figure 4: The tope layer

$$\frac{\Xi \mid \Phi \vdash \bot}{\Xi \mid \Phi \mid \Gamma \vdash \mathsf{rec}_\bot : A} \qquad \frac{\Xi \mid \Phi \vdash \bot \qquad \Xi \mid \Phi \mid \Gamma \vdash a : A}{\Xi \mid \Phi \mid \Gamma \vdash a \equiv \mathsf{rec}_\bot}$$

$$\frac{\Xi \mid \Phi \vdash \phi \vee \psi \qquad \Xi \mid \Phi \mid \Gamma \vdash A\ \mathsf{type} \qquad \Xi \mid \Phi, \phi \mid \Gamma \vdash a_\phi : A \qquad \Xi \mid \Phi, \psi \mid \Gamma \vdash a_\psi : A \qquad \Xi \mid \Phi, \phi \wedge \psi \mid \Gamma \vdash a_\phi \equiv a_\psi}{\Xi \mid \Phi \mid \Gamma \vdash \mathsf{rec}_\vee^{\phi,\psi}(a_\phi, a_\psi) : A}$$

$$\frac{\Xi \mid \Phi \vdash \phi \vee \psi \qquad \Xi \mid \Phi \mid \Gamma \vdash A\ \mathsf{type} \qquad \Xi \mid \Phi, \phi \mid \Gamma \vdash a_\phi : A \qquad \Xi \mid \Phi, \psi \mid \Gamma \vdash a_\psi : A \qquad \Xi \mid \Phi, \phi \wedge \psi \mid \Gamma \vdash a_\phi \equiv a_\psi}{\Xi \mid \Phi, \phi \mid \Gamma \vdash \mathsf{rec}_\vee^{\phi,\psi}(a_\phi, a_\psi) \equiv a_\phi}$$

$$\frac{\Xi \mid \Phi \vdash \phi \vee \psi \qquad \Xi \mid \Phi \mid \Gamma \vdash A\ \mathsf{type} \qquad \Xi \mid \Phi, \phi \mid \Gamma \vdash a_\phi : A \qquad \Xi \mid \Phi, \psi \mid \Gamma \vdash a_\psi : A \qquad \Xi \mid \Phi, \phi \wedge \psi \mid \Gamma \vdash a_\phi \equiv a_\psi}{\Xi \mid \Phi, \psi \mid \Gamma \vdash \mathsf{rec}_\vee^{\phi,\psi}(a_\phi, a_\psi) \equiv a_\psi}$$

$$\frac{\Xi \mid \Phi \vdash \phi \vee \psi \qquad \Xi \mid \Phi \mid \Gamma \vdash a : A}{\Xi \mid \Phi \mid \Gamma \vdash a \equiv \mathsf{rec}_\vee^{\phi,\psi}(a, a)}$$

Figure 5: Type elimination for tope disjunction

$$\begin{aligned}
x :\nvDash \mid \cdot &\vdash (x \leq x) \\
x :\nvDash, y :\nvDash, z :\nvDash \mid (x \leq y), (y \leq z) &\vdash (x \leq z) \\
x :\nvDash, y :\nvDash \mid (x \leq y), (y \leq x) &\vdash (x \equiv y) \\
x :\nvDash, y :\nvDash \mid \cdot &\vdash (x \leq y) \vee (y \leq x) \\
x :\nvDash \mid \cdot &\vdash (0 \leq x) \\
x :\nvDash \mid \cdot &\vdash (x \leq 1) \\
\cdot \mid (0 \equiv 1) &\vdash \bot
\end{aligned}$$

Figure 6: Strict interval axioms

$$\frac{\begin{array}{c}\{t : I \mid \phi\}\ \mathsf{shape} \qquad \{t : I \mid \psi\}\ \mathsf{shape} \qquad t : I \mid \phi \vdash \psi \\ \Xi \mid \Phi \vdash \Gamma\ \mathsf{ctx} \qquad \Xi, t : I \mid \Phi, \psi \mid \Gamma \vdash A\ \mathsf{type} \qquad \Xi, t : I \mid \Phi, \phi \mid \Gamma \vdash a : A\end{array}}{\Xi \mid \Phi \mid \Gamma \vdash \left\langle \prod_{t:I \mid \psi} A \middle| {}^{\phi}_{a} \right\rangle\ \mathsf{type}}$$

$$\frac{\begin{array}{c}\{t : I \mid \phi\}\ \mathsf{shape} \qquad \{t : I \mid \psi\}\ \mathsf{shape} \qquad t : I \mid \phi \vdash \psi \\ \Xi \mid \Phi \vdash \Gamma\ \mathsf{ctx} \qquad \Xi, t : I \mid \Phi, \psi \mid \Gamma \vdash A\ \mathsf{type} \qquad \Xi, t : I \mid \Phi, \phi \mid \Gamma \vdash a : A \\ \Xi, t : I \mid \Phi, \psi \mid \Gamma \vdash b : A \qquad \Xi, t : I \mid \Phi, \phi \mid \Gamma \vdash b \equiv a\end{array}}{\Xi \mid \Phi \mid \Gamma \vdash \lambda t^{I \mid \psi}.\, b : \left\langle \prod_{t:I \mid \psi} A \middle| {}^{\phi}_{a} \right\rangle}$$

$$\frac{\begin{array}{c}\{t : I \mid \phi\}\ \mathsf{shape} \qquad \{t : I \mid \psi\}\ \mathsf{shape} \qquad t : I \mid \phi \vdash \psi \\ \Xi \mid \Phi \mid \Gamma \vdash f : \left\langle \prod_{t:I \mid \psi} A \middle| {}^{\phi}_{a} \right\rangle \qquad \Xi \vdash s : I \qquad \Xi \mid \Phi \vdash \psi[s/t]\end{array}}{\Xi \mid \Phi \mid \Gamma \vdash f(s) : A}$$

$$\frac{\begin{array}{c}\{t : I \mid \phi\}\ \mathsf{shape} \qquad \{t : I \mid \psi\}\ \mathsf{shape} \qquad t : I \mid \phi \vdash \psi \\ \Xi \mid \Phi \mid \Gamma \vdash f : \left\langle \prod_{t:I \mid \psi} A \middle| {}^{\phi}_{a} \right\rangle \qquad \Xi \vdash s : I \qquad \Xi \mid \Phi \vdash \phi[s/t]\end{array}}{\Xi \mid \Phi \mid \Gamma \vdash f(s) \equiv a[s/t]}$$

$$\frac{\begin{array}{c}\{t : I \mid \phi\}\ \mathsf{shape} \qquad \{t : I \mid \psi\}\ \mathsf{shape} \qquad t : I \mid \phi \vdash \psi \\ \Xi \mid \Phi \vdash \Gamma\ \mathsf{ctx} \qquad \Xi, t : I \mid \Phi, \psi \mid \Gamma \vdash A\ \mathsf{type} \qquad \Xi, t : I \mid \Phi, \phi \mid \Gamma \vdash a : A \\ \Xi, t : I \mid \Phi, \psi \mid \Gamma \vdash b : A \\ \Xi, t : I \mid \Phi, \phi \mid \Gamma \vdash b \equiv a \qquad \Xi \vdash s : I \qquad \Xi \mid \Phi \vdash \psi[s/t]\end{array}}{\Xi \mid \Phi \mid \Gamma \vdash (\lambda t^{I \mid \psi}.\, b)(s) \equiv b[s/t]}$$

$$\frac{\begin{array}{c}\{t : I \mid \phi\}\ \mathsf{shape} \qquad \{t : I \mid \psi\}\ \mathsf{shape} \qquad t : I \mid \phi \vdash \psi \\ \Xi \mid \Phi \mid \Gamma \vdash f : \left\langle \prod_{t:I \mid \psi} A \middle| {}^{\phi}_{a} \right\rangle\end{array}}{\Xi \mid \Phi \mid \Gamma \vdash f \equiv \lambda t^{I \mid \psi}.\, f(t)}$$

Figure 7: Extension types

References

[AW09] STEVE AWODEY and MICHAEL A. WARREN. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1):45–55, January 2009.

[BW22] Ulrik Buchholtz and Jonathan Weinberger. Synthetic fibered $(\infty,1)$-category theory, August 2022. arXiv:2105.01724.

[DR11] Daniel R. Licata and Robert Harper. 2-Dimensional Directed Type Theory. *Electronic Notes in Theoretical Computer Science*, 276:263–289, September 2011.

[GG08] Nicola Gambino and Richard Garner. The identity type weak factorisation system. *Theoretical Computer Science*, 409(1):94–109, December 2008.

[GWB24] Daniel Gratzer, Jonathan Weinberger, and Ulrik Buchholtz. Directed univalence in simplicial homotopy type theory, July 2024. arXiv:2407.09146 [cs].

[HS98] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. *Twenty-five years of constructive type theory (Venice, 1995)*, 36:83–111, 1998.

[KL18] Chris Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of univalent foundations (after voevodsky), 2018.

[KRW23] Nikolai Kudasov, Emily Riehl, and Jonathan Weinberger. Formalizing the ∞-Categorical Yoneda Lemma, December 2023. arXiv:2309.08340.

[LS13] Daniel R. Licata and Michael Shulman. Calculating the fundamental group of the circle in homotopy type theory. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '13, page 223–232, USA, 2013. IEEE Computer Society.

[Nor19] Paige Randall North. Towards a Directed Homotopy Type Theory. *Electronic Notes in Theoretical Computer Science*, 347:223–239, November 2019.

[Nuy15] Andreas Nuyts. *Towards a Directed Homotopy Type Theory based on 4 Kinds of Variance.(2015)*. PhD thesis, Master thesis, available online at https://distrinet. cs. kuleuven. be . . . , 2015.

[Pro13] The Univalent Foundations Program. Homotopy Type Theory: Univalent Foundations of Mathematics, August 2013. arXiv:1308.0729.

[Rij22] Egbert Rijke. Introduction to homotopy type theory, 2022.

[RS17] Emily Riehl and Michael Shulman. A type theory for synthetic ∞-categories. *Higher Structures*, 1(1):147–224, December 2017.

[vdBG10] Benno van den Berg and Richard Garner. Types are weak ω-groupoids. *Proceedings of the London Mathematical Society*, 102(2):370–394, 10 2010.

[Wad89] Philip Wadler. Theorems for free! In *Proceedings of the fourth international conference on Functional programming languages and computer architecture*, pages 347–359, 1989.

[War08] M Warren. Homotopy theoretic aspects of constructive type theory. 2008.