Bicubical Directed Type Theory

Matthew Zachary Weaver

A Dissertation

Presented to the Faculty

of Princeton University

in Candidacy for the Degree

of Doctor of Philosophy

Recommended for Acceptance

by the Department of

Computer Science

Advisers: Daniel R. Licata & Andrew W. Appel

November 2024

Abstract

In homotopy type theory, each type is equipped with an abstract notion of path, corresponding to the morphisms of a ∞-groupoid. Voevodsky's univalence axiom states that the type of paths between types is equivalent to the type of equivalences between types, and can be seen as extending type theory with a generic program for lifting equivalences between types along any type family. Defining this lifting for all dependent types is quite subtle, but univalence has now been given a computational interpretation in various cubical type theories.

A natural question is whether there are any directed analogues of homotopy type theory, where types are equipped with a notion of directed morphism and correspond to higher categories, generalizing groupoids. In such a setting, one possible directed analogue of univalence would circumscribe a class of type constructors that represent covariant functors, and provide a means to automatically lift a function or directed morphism along such a type constructor. Directed type theory with directed univalence has some potential applications to computer science that are not possible in ordinary homotopy type theory, such as providing a convenient language for coding the functorial specification of abstract syntax.

One approach to directed type theory, developed by Riehl and Shulman, is based on equipping each type with both a notion of path and a separate notion of directed morphism [62]. While ordinary homotopy type theory has models in simplicial sets, the Riehl-Shulman type theory is modeled in bisimplicial sets, capturing these two notions of path and directed morphism. While this suffices for formalizing mathematics, for applications to computer science we would like a computational interpretation of the type theory, and it is not yet known whether there are constructive models of homotopy type theory in simplicial sets.

Thus we instead consider a cubical setting, and give a constructive model of a directed type theory with directed univalence in bicubical sets. We formalize much of this using Agda as the internal language of a topos, following Orton and Pitts [58].

First, building on the cubical techniques used to give computational models of homotopy type theory, we show that there is a universe of covariant discrete fibrations with a partial directed univalence principle asserting that functions are a retract of morphisms in this universe. To complete this retraction into an equivalence, we refine the model using Coquand, Ruch, and Sattler's work on constructive sheaf models [28]. We introduce the cobar modality and by restricting to fibrant types that are also cobar modal, we are able to complete our construction of directed univalence.

We then describe a generalization of the fibrant Riehl-Shulman extension types defined in [62]. We prove this in the setting of an arbitrary presheaf category with respect to a new notion of fibrancy that is given by a generic filling problem. This abstraction is general enough to capture all of the current presheaf models of type theories and their classifications of types specified by filling problems. In addition, this result extends the potential syntax of these type theories to be able to internally express any of these filling problems as fibrant types. We use this to then define a type theory in which the user can internally define classifying universes for any such filling problem.

Lastly, we overview our implementation of bicubical directed type theory focusing on a few interesting design decisions in regards to its syntax. As opposed to exposing the connections for the directed interval we chose to instead include inequality cofibrations directly in the syntax and omit connections, resulting in a syntax that is sufficiently expressive while providing substantial benefits computationally.

Acknowledgements

I would like to thank the incredible number of people that helped me through this grueling but rewarding journey.

My time at Penn as an undergraduate was a wonderful and enjoyable time in my life, and everything I learned from the PL Club and my undergraduate research advisor Stephanie Weirich ensured I was prepared for my time in graduate school. In a similar vein, I greatly appreciate the Princeton PL Group and how it was a welcoming and supportive community, making my time in the office consistently fun.

I would like to thank Carlo Angiuli, Steve Awodey, Evan Cavallo, Thierry Coquand, Favonia, Bob Harper, Anders Mortbërg, Reed Mullanix, Emily Riehl, Mitchell Riley, Robert Rose, Christian Sattler, Jon Sterling, Dimitris Tsementzis, Jonathan Weinberger and everyone else who collaborated and provided feedback throughout the entirety of my PhD.

Thank you Zak Kincaid and Dave Walker for serving on my committee.

I also thank Andrew Appel for doing so much for me throughout my years at Princeton; from ensuring I kept one eye grounded in applications despite my work living in the abstract clouds to forcing me to finally finish my dissertation and graduate, your support made this thesis possible.

Finally, I thank Daniel Licata for truly being the most encouraging advisor I could ever ask for.

This material is based on research sponsored by The United States Air Force Research Laboratory under agreement numbers FA9550-16-1-0292, and FA9550-21-0009 (Tristan Nguyen, program manager). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the United States Air Force Research Laboratory, the U.S. Government.

To my mom and dad for their endless support

Contents

List of Figures

Chapter 1

Introduction

In homotopy type theory [11, 74, 76], each type is equipped with an abstract notion of path, corresponding to the morphisms of a higher groupoid. Voevodsky's univalence axiom states that the type of paths between types is equivalent (a generalization of isomorphic) to the type of equivalences between types. This can be seen as extending type theory with a generic program for lifting equivalences between types along any type family: given an equivalence $A \simeq B$ and any family of types $C : U \to U$, one can automatically construct an equivalence $C(A) \simeq C(B)$. Defining this lifting for all dependent types is quite subtle, but univalence has been given a computational interpretation via cubical type theories [7, 10, 13, 24].

A natural question is whether there are any *directed* analogues of homotopy type theory, where types are equipped with a notion of directed morphism and correspond to higher categories, generalizing groupoids. In such a setting, one possible directed analogue of univalence would circumscribe a class of type constructors that represent covariant functors, and provide a means to automatically lift a *function* $A \to B$ to a function $C(A) \to C(B)$, generalizing e.g. the Functor type class of Haskell to dependent types. By analogy with univalence, such a lifting would result from a universe classifying covariant type families that satisfies a *directed univalence* principle stating that the directed morphisms in this universe are equivalent to functions. On the mathematical side, directed type theory provides a more

general language for synthetic mathematics. On the computational side, one potential application is to defining abstract syntax [31, 41, 45]; others might arise from the connections between directed homotopy theory and concurrency [30, 37, 52].

1.1 A Motivating Example

The benefits of using directed type theory for the formal verification of computation structures are quite promising. Beyond the internal categorical structures providing a concise way to package the information and enforce invariants, one can also leverage the abstract results from category theory to significantly decrease the amount of work required to accomplish the same results when compared to existing verification approaches. One notable instance of this is taking advantage of the fact that the universe is closed under many type formers.

For example, given type families A and B in $C \to \mathsf{U}$, one can always always define the type family $A \times B : C \to \mathsf{U}$, and its directed structure is uniquely determined from A and B. Thus, the user of such a type theory must provide the additional directed structure when there is nontrivial information in the definition of the type theory, but having done so one can then build up larger types that inherit the directed structure via universal properties without the user needing to provide anything extra.

Before the technical details, we sketch a motivating example of defining abstract syntax in a directed type theory, building on [31, 41, 45]. This sketch will use a directed higher inductive type, which we do not formally study in this thesis. We write $\mathsf{Hom}_A(x, y)$ for the directed morphism type, representing a morphism from x to y—this should be thought of like the $\mathsf{Path}_A(x, y)$ or identity $\mathsf{Id}_A(x, y)$ type of homotopy type theory, except in general we always have identity morphisms for every type, but not necessarily composition or inverses. Those types A whose morphisms have (homotopy) unique composites (and so correspond to categories) are called *Segal* [62].

For this example, we will define the syntax of the untyped lambda calculus with scoped de Bruijn variables. By doing so within directed type theory, we can encode the weakening substitution that introduces a new variable to the scope as an intrinsic component to the structure of the types we define; furthermore, by solely providing the function that describes how to weaken variables in the context, the type theory automatically identifies the unique function that soundly weakens lambda terms without any additional input from the user.

The first idea in our example is to represent object-language contexts by a type whose elements represent contexts, and whose morphisms represent object-language substitutions or renamings. As a concrete example, we use the untyped lambda calculus, indexed by contexts denoting how many free variables are present, and take the substitutions to be weakenings only. The type of contexts should be defined as a Segal directed higher inductive type:

```
data Ctx : USegal where
  empt : Ctx
  ext  : Ctx → Ctx
  wk   : (G : Ctx) → Hom Ctx G (ext G)
```

The general idea for the above definition is that we have directed higher inductive types based on the syntax for higher inductive types described in [19]; the definition is "directed" as the wk constructor generates a morphism in $\mathsf{Hom}\ \mathsf{Ctx}\ \mathsf{G}\ (\mathsf{ext}\ \mathsf{G})$ as opposed to a path. This declaration is intended to mean that the elements of Ctx are in bijection with the natural numbers, with directed morphisms generated by $\mathsf{wk}_n : \mathsf{Hom}_{\mathsf{Ctx}}(n, n+1)$ corresponding to weakening. We declare Ctx to be Segal by defining it as an element of the universe $\mathsf{U}_{\mathsf{Segal}}$. This indicates that it represents a category and thus must have freely generated composites; e.g. $\mathsf{wk}_1 \circ \mathsf{wk}_0 : \mathsf{Hom}_{\mathsf{Ctx}}(0, 2)$ represents weakening the empty context with two variables. Because functions such as ext can also act on morphisms, we also have morphisms like $\mathsf{ext}(\mathsf{wk}_0) : \mathsf{Hom}_{\mathsf{Ctx}}(1, 2)$ that weakens in the middle of a context.

We then define a type family $\mathsf{Var} : \mathsf{Ctx} \to \mathsf{U}_{\mathsf{cov}}$ representing the variables in each context. Before doing so, let us first inspect the type of Var: The type theory automatically enforces

that all functions preserve morphisms, and thus in addition to providing a type of "variables in context G for each context G," the type family Var takes morphisms in Ctx (i.e. weakenings) to morphisms in the universe $\mathsf{U}_{\mathsf{cov}}$; the fact that Var lands in the covariant universe $\mathsf{U}_{\mathsf{cov}}$ in particular indicates that these morphisms between types in the universe are given by actual functions between the types at each end of the morphism as a result of directed univalence. For this to all workout, a particular transport function from n to $n+1$ is given in the definition of Var for the weakening axiom—the inl injection—which is turned into a morphism in UCov using directed univalence. The elimination form for a Segal higher inductive type would then send compositions of weakenings to compositions of these injections. As a result, the type family Var automatically includes the code for shifting a de Bruijn index.

```
Var : Ctx → UCov
Var empt    = 0
Var (ext G) = (Var G) + 1
Var (wk G)  = dua inl


data Tm (G : Ctx) : UCov where
  var : Var G → Tm G
  abs : Tm (ext G) → Tm G
  app : Tm G → Tm G → Tm G
```

Finally, we define a type family of lambda terms $\mathsf{Tm} : \mathsf{Ctx} \to \mathsf{U}_{\mathsf{cov}}$ that is constructed from a base case Var that maps into $\mathsf{U}_{\mathsf{cov}}$, and otherwise is a sum of products of inductive occurrences. Because $\mathsf{U}_{\mathsf{cov}}$ is closed under products, sums, and (conjecturally) certain inductive types, Tm will *automatically* land in $\mathsf{U}_{\mathsf{cov}}$. This means that, while the definition of Tm looks identical to what one would write in standard type theory, the usual code for lifting weakening to terms is written automatically by the transport functions for these types. We use this by

```
dtransport Tm (wk G) : Tm G → Tm (ext G)
```

to, for example, weaken a term with one extra variable, where $\mathsf{dtransport}$ is an operation defined for any type family that lands in $\mathsf{U}_{\mathsf{cov}}$. General facts about directed transport will imply

that these substitution actions commute with composition, which is needed in metatheory and in defining typing judgments, program logics, or dependent types.

For this example, we need a universe $\mathsf{U}_{\mathsf{cov}}$ (Sections 3.3 and 4.2.1) with directed univalence (Chapter 4), closed under products, sums, and inductive types, which is itself a category (*Segal,* Section 3.2.4 and [62]); and we also need directed higher inductive types. Of these, we leave the closure of $\mathsf{U}_{\mathsf{cov}}$ under inductive types and the mechanism of directed higher inductive types to future work, and focus on defining a universe $\mathsf{U}_{\mathsf{cov}}$ supporting directed univalence.

1.2 A Subjective Treatise on Research in Type Theory

As a necessary disclaimer—despite the punchy title of this section—this is not an attempt to define what is "good" or "correct" research in type theory, and certainly is far from a comprehensive perspective on the subject; the description below is simply the framework I have maintained in my head throughout the development of this work. I hope by explaining how I personally organize my thoughts when approaching my research, you as the reader might gain some useful insights that will improve your own understanding of the content that is to follow throughout the remainder of my thesis.

Defining a type theory, in particular one of a more theoretical, categorical nature, tends to follow a common structure: 1. add new structure to the semantics of types beyond them simply being sets; 2. add syntax to the theory allowing one to describe and interact with the additional structure; 3. define a property classifying which types are the "good," "well-behaved" types with respect to this additional structure; 4. define a universe classifying the types that respect the property; 5. derive a universal theorem classifying the new structure for the universe that fundamentally leverages the property of types that the universe classifies. Let's pull this apart and go a bit deeper into what I mean by this dense summary of the framework.

Semantic Structure

The first step in this style of research is to select a semantic setting for ones types. Of course, typically one has a goal of what types should represent and with it a candidate in mind, but the key observation is that the type theory will enrich the notion of type beyond simply a set of terms. In this thesis, our types are semantically given by bicubical sets with the goal of types representing ∞-categories, but to explain this framework let us consider a much more tangible example: quotients of sets.

Imagine we are trying to define a type theory that finally handles quotients perfectly, saving us once and for all from "setoid hell."[1] Naïvely, how might we start? As everything leading up to this paragraph has anticipated, we first choose to enrich our notion of types! Instead of semantically interpreting our types as sets, let us instead choose to interpret them as a set paired with a relation: using type theoretic inspired notation, each type is represented as a pair (A, R) of the form $\Sigma A : \mathbf{Set}.\mathcal{P}(A \times A)$. A is the set representing the terms of the type, and $R \subseteq A \times A$ is the relation indicating which terms are equated in the quotient. With this, we have selected a semantic domain and thus now have the mathematical setting in which we will build up our new type theory.

To help make these ideas more concrete, we will consider the type of untyped lambda terms as a running example throughout this treatise. First, let us look at how one could represent the semantic representation of the type of named lambda calculus terms with the relation representing alpha-equivalence. We will use $\mathbb{N}$ to denote the countably infinite set of names, conveniently coinciding with the notation of natural numbers. The set of named lambda terms $\mathbf{LC}$ is then given as the least fixed point of the function $F\ X = \mathbb{N} + X \times X + \mathbb{N} \times X$. While this may give a formal mathematical definition of the set, its

[1] The term "setoid hell" has been used to describe the challenges that traditionally occur when trying to program or write mechanized proofs involving quotients. To represent quotients one uses setoids: types equipped with a binary relation representing which terms are equated by the quotient. When working with setoids, one must prove that every function preserves the setoid relation and it quickly becomes very tedious and exhausting, motivating the apt description of "setoid hell."

contents become more clear if we give the names **var**, **app** and **abs** to the three inclusions and make explicit the result of the least fixed point.

$$\begin{aligned} \mathbf{var}\ x &\in \mathbf{LC} && \text{for all } x \in \mathbb{N} \\ \mathbf{app}\ f\ t &\in \mathbf{LC} && \text{for all } f \in \mathbf{LC} \text{ and } t \in \mathbf{LC} \\ \mathbf{abs}\ x\ t &\in \mathbf{LC} && \text{for all } x \in \mathbb{N} \text{ and } t \in \mathbf{LC} \end{aligned}$$

Having defined the set, we now must equip it with a relation $R_{\mathbf{LC}} \subseteq \mathbf{LC} \times \mathbf{LC}$. We do so with the following rules.

$$\begin{aligned} (t, t) &\in R_{\mathbf{LC}} && \text{for all } t \in \mathbf{LC} \\ (\mathbf{app}\ f\ t, \mathbf{app}\ f'\ t) &\in R_{\mathbf{LC}} && \text{for all } (f, f') \in R_{\mathbf{LC}} \text{ and } t \in \mathbf{LC} \\ (\mathbf{app}\ f\ t, \mathbf{app}\ f\ t') &\in R_{\mathbf{LC}} && \text{for all } f \in \mathbf{LC} \text{ and } (t, t') \in R_{\mathbf{LC}} \\ (\mathbf{abs}\ x\ t, \mathbf{abs}\ x\ t') &\in R_{\mathbf{LC}} && \text{for all } x \in \mathbb{N} \text{ and } (t, t') \in R_{\mathbf{LC}} \\ (\mathbf{abs}\ x\ t, \mathbf{abs}\ y\ t[\mathbf{var}\ y/x]) &\in R_{\mathbf{LC}} && \text{for all } y \in \mathbb{N} \text{ fresh in } t \in \mathbf{LC} \end{aligned}$$

In the above rules, the notation $t[t'/x]$ represents the term resulting from substituting the term t' for the variable x in the term t. We also use the standard definition for when a variable is fresh in a lambda term. Combining the two definitions, we now have defined the terms of the lambda calculus up to alpha-equivalence within our semantic domain as the pair $(\mathbf{LC}, R_{\mathbf{LC}})$.

Structural Syntax

Adding structure to the semantic interpretation of types begins the process of defining a new types theory, but in actuality doesn't change much. Assuming it is an appropriate domain (i.e. a topos, see Section 2.2.3), we know it supports dependent type theory, and thus automatically the theory contains all of the type formers and structure we are accustomed to having. So what is the problem? While the new structure is semantically present, it is invisible to the type theory and thus doing absolutely nothing; we may as well interpret the type theory into sets. To fix this problem, we must extend the syntax with new primitives that provide the theory access to the structure we have added.

What syntactic elements might we want in quotient type theory? For starters, we probably would want a way to project out the relation of any given type and be able to ask

whether two terms are related with respect to the relation. With that in mind, we add the following formation rule:

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma \vdash a_0 : A \qquad \Gamma \vdash a_1 : A}{\Gamma \vdash a_0 \sim_A a_1 : \mathsf{Type}}$$

Given two terms a_0 and a_1 in A, we have a type representing the proposition of whether a_0 is related to a_1 by the relation. We also likely want to be able to be able to get an answer to whether the two terms are related, and thus should add an introduction rule as well. As this is imaginary, we will not worry about any computational challenges one may have about decidability. We also will not worry about the precise semantic interpretation of the syntax, and just assume we have a perfect interpretation function $[\![-]\!]$ that translates any syntax we need into the semantic domain. With these *aspirational* assumptions and use of hand-waving, we can add the following introduction rule to our type theory:

$$\frac{([\![a_0]\!], [\![a_1]\!]) \in \pi_2([\![A]\!])}{\Gamma \vdash * : a_0 \sim_A a_1}$$

If two terms are related semantically we have a syntactic witness to the proposition asking whether they are related. Of course, were we defining a real type theory we would need this to be precise and likely want to expose more structure in a more intelligent way, but the point of this example is to provide intuition on the research framework and not actually do novel research, and so we continue to the third step.

Returning to our example, assume we have a type LC of lambda terms such that $[\![\mathsf{LC}]\!] = \mathbf{LC}$. Also assume the terms of LC are built from constructors $\mathsf{var} : \mathsf{Nat} \to \mathsf{LC}$, $\mathsf{app} : \mathsf{LC} \to \mathsf{LC} \to \mathsf{LC}$, and $\mathsf{abs} : \mathsf{Nat} \to \mathsf{LC} \to \mathsf{LC}$ (where Nat is the type of natural numbers) that interpret to align with the named inclusions **var**, **app** **abs** from the semantic domain. Based upon the above rules, we always have a witness $* : t \sim_{\mathsf{LC}} t$ for ever term $t : \mathsf{LC}$, there is a witness $* : \mathsf{app}\ f\ t \sim_{\mathsf{LC}} \mathsf{app}\ f\ t'$ whenever we have a witness to $t \sim_{\mathsf{LC}} t'$, and so on. Given how we defined the witness term for relations, note that the type theory is assuming an oracle

can always solve whether two terms are related in order to type-check the witness, and in the case of alpha-equivalence for lambda terms we know it is relatively straightforward to define such an oracle algorithmically. Given this additional syntax, the type theory can now talk about whether or not two lambda terms represented within LC are alpha-equivalent as the relation is now accessible to the surface syntax. Making this fully concrete, the type theory "knows" the following types are inhabited.

$$\begin{array}{rcl} \mathsf{abs}\ 0\ 0 & \sim_{\mathsf{LC}} & \mathsf{abs}\ 1\ 1 \\ \mathsf{app}\ (\mathsf{abs}\ 0\ 1)\ 2 & \sim_{\mathsf{LC}} & \mathsf{app}\ (\mathsf{abs}\ 3\ 1)\ 2 \\ \mathsf{abs}\ 0(\mathsf{app1}\ 0) & \sim_{\mathsf{LC}} & \mathsf{abs}\ 2\ (\mathsf{app}\ 1\ 2) \end{array}$$

The Property

At this point the type theory has additional structure and the structure is even visible to a user of the theory. What else is required before calling the project complete? In practice, the structure that is most convenient or natural to add to the types is often too general for their desired interpretation, and thus there are many types semantically we wish to exclude from the theory. The types we are actually interested in—the goal of what types should represent in this new theory—are those types that additionally satisfy a property pertaining to the new structure. Given this is the case, we then must identify the property and restrict our type theory to those satisfying the structure; in doing so, we must also show that every type former we wish to include preserves the property so that we cannot ever create a type that no longer satisfies the property.

The observant reader may have already identified the property missing from our quotient type theory before reaching this point: To define a type theory of quotiented sets, we need types to not just be pairs of a set with an arbitrary relation but specifically those sets that are paired with an *equivalence* relation. To spell this out, we restrict to types A that satisfy the following three predicates expressed syntactically within the type theory:

$$\begin{array}{rcl} \mathsf{isRefl}\ A & : & \Pi a : A.\ a \sim_A a \\ \mathsf{isSymm}\ A & : & \Pi a_0\ a_1 : A.\ a_0 \sim_A a_1 \to a_1 \sim_A a_0 \\ \mathsf{isTrans}\ A & : & \Pi a_0\ a_1\ a_2 : A.\ a_0 \sim_A a_1 \to a_1 \sim_A a_2 \to a_0 \sim_A a_2 \end{array}$$

The types we restrict our theory to are those whose relations are reflexive, symmetric and transitive.

Definition 1.2.1. A type A equipped with witnesses of the above predicates indicating its binary relation is reflexive, symmetric and transitive is called an *equivalence type*.

A key aspect of this step is that, while we define this restriction on types syntactically in the logic, it also interprets into a meaningful property semantically. Expanding the above syntactic restriction definitionally, we see that the rules interpret to the following statements.

$$\begin{array}{rcl} [\![\mathsf{isRefl}\ A]\!] & : & \forall a \in \pi_1([\![A]\!]),\ (a,a) \in \pi_2([\![A]\!]) \\ [\![\mathsf{isSymm}\ A]\!] & : & \forall a_0,\ a_1 \in \pi_1([\![A]\!]), (\ a_0, a_1) \in \pi_2([\![A]\!]) \Rightarrow (a_1, a_0) \in \pi_2([\![A]\!]) \\ [\![\mathsf{isTrans}\ A]\!] & : & \forall a_0,\ a_1,\ a_2 \in \pi_1([\![A]\!]), (a_0, a_1) \in \pi_2([\![A]\!]) \text{ and } (a_1, a_2) \in \pi_2([\![A]\!]) \\ & & \Rightarrow (a_0, a_2) \in \pi_2([\![A]\!]) \end{array}$$

Looking at the above interpretations, we can see that by definition we restrict to types A such that $\pi_2([\![A]\!])$ is an equivalence relation over the set $\pi_1([\![A]\!])$. In setting up the theory with such a clear connection to semantic domain with a direct interpretation of the property, we make it easy to ultimately interpret definitions and proofs given within the type theory into the semantics, allowing one to extract any type theoretic results for safe usage with the mathematics of the semantic domain at large, in our case here being sets equipped with an equivalence relation.

To ensure we can do this restriction and still use our type formers without worry, we then must prove lemmas such as,

Lemma 1.2.2. *For every equivalence type A and every type family $B : A \to \mathsf{Type}$ such that all types $B\ a$ in the family are equivalence types, the type $\Sigma a : A.B\ a$ is also an equivalence type.*

As the novel type theory we currently are defining is not real, we shall skip writing such proofs and simply assume they exist.

Returning to our example of the lambda calculus, anyone following this approach will have to have proven that the notion of alpha-equivalence is indeed reflexive, symmetric and transitive, and thus we can safely add witnesses for $\mathsf{isRefl\ LC}$, $\mathsf{isSymm\ LC}$ and $\mathsf{isTrans\ LC}$ to the syntax concluding that LC is an equivalence type in this theory.

The Universe

The penultimate step of this research framework is to collect all of our well-behaved types into a common universe. While initially this may seem nearly trivial, doing so in such a way that is constructive can be quite challenging depending on the semantic setting and the choice of property that defines which types are well-behaved. A secondary task that commonly accompanies this step is then showing that the universe itself satisfies the property, allowing us to create a "tower" of universes (i.e. $\mathsf{U}_i : \mathsf{U}_{i+1}$), and in doing so providing a solution to any potential size issues. At this point, assuming all type former preserve the property, we have finally arrived at a type theory that can syntactically forget about ill-behaved types along with the fully-general universe Type containing both the "good" and "bad" types; we have enough syntactic justification to define the entire theory with respect to our new (tower of) universe(s) $\mathsf{U}_{(i)}$, and from our previously proven lemmas we know all reachable syntax remains within our intended setting of types with the additional structure that also satisfy the property.

In line with our increasingly casual treatment of quotient type theory up to this point, let's assume we can define a universe $\mathsf{U}_{\mathsf{eqv}}$ that contains all of the equivalence types. Given we have assumed the presence of proofs stating that every type former imaginable preserves equivalence relations, we know it is inhabited by desired types, including the running example of lambda terms LC. Let us also assume that $\mathsf{U}_{\mathsf{eqv}}$'s relation is also an equivalence relation, and thus we can safely define a Grothendieck tower of universes. We have solved setoid hell! That being said, there is still one step remaining before this result is complete.

The Universal Property

While at this point we have a fully functioning type theory with all of the additional structure we intended, one open question remains: What does this additional structure look like for the universe? As every type in the type theory is enriched with the structure we set out to add, the universe is no exception; however, there is no indication within the type theory of what the structure is doing, and of whether or not it is useful for a user of the theory. All we know is that the universe has the structure and that said structure on the universe satisfies the property indicating it is well-behaved. To ameliorate this, the final step is to classify the structure of the universe with a general theorem. Ideally it connects the structure in the universe to a more traditional type, often along an equivalence or something similar. The canonical example of doing this is univalence in homotopy type theory: For any two types A and B, the type of paths in the universe between A and B is equivalent to the type of equivalences between A and B.

$$\mathsf{Path}_{\mathsf{U}}\,(A, B) \simeq (A \simeq B)$$

The property translates the structure we have added to the universe into something tangible that can be leveraged inside the type theory. This is generally the most challenging step of the research project, but also the most rewarding as it typically provides a very powerful tool that fundamentally motivates adding all of the new structure to the notion of what it means to be a type in the first place.

What might the equivalence relation on $\mathsf{U}_{\mathsf{eqv}}$ be? To answer this question, we will use the following definition of type equivalence.

Definition 1.2.3. Given types A and B, the type of *equivalences* between A and B, written $A \simeq B$ contains pairs (f, g) where f is a function $A \to B$ and g a function $B \to A$ such that for every $a : A$, we have a witness that $a \sim_A (g \circ f)\ a$ and for every $b : B$ we have a witness that $b \sim_B (f \circ g)\ b$.

For the sake of most naturally aligning with the real and proven results that will follow in this thesis, let us claim that it satisfies the following notion of *quotient univalence*: For any two types A and B in $\mathsf{U}_{\mathsf{eqv}}$, the type asking whether A is related to B in $\mathsf{U}_{\mathsf{eqv}}$ is equivalent that asking whether A is equivalent B.

$$\cdot \vdash \mathsf{qua} : \Pi A\ B : \mathsf{U}_{\mathsf{eqv}}.(A \sim_{\mathsf{U}_{\mathsf{eqv}}} B) \simeq (A \simeq B)$$

Note that we can use quotient univalence on itself to conclude a stronger version: $(A \sim_{\mathsf{U}_{\mathsf{eqv}}} B) \sim_{\mathsf{U}_{\mathsf{eqv}}} (A \simeq B)$. The quotient relation of the universe coincides with the notion of equivalence of types. We have now concluded our work on quotient type theory!

Before leaving this example behind, let us think a bit more on why we even cared to prove quotient univalence. Obviously our sketch of quotient type theory was not mathematically sound in the least and missing *many* details; one particular detail missing is that—in our perfect imaginary world void of setoid hell—all of syntactic constructions (and in particular functions) would not only preserve the property of being an equivalence relation, but specifically the structure of the relations themselves; this is necessary as we certainly need our quotient type theory to actually respect quotients, and thus functions must send related terms in the domain to related terms in the codomain. An important implication of this follows from the observation that, when constructing a type in the universe, often one does so in a non-empty context that contains types with non-trivial quotient relations (i.e. dependent types exist). Upon defining such a type, which we will wrap up neatly with the context into a type family $B : A \to \mathsf{U}_{\mathsf{eqv}}$, we can automatically use the fact that functions respect equivalence relations to conclude that whenever we have a proof of $a_0 \sim_A a_1$ we also have a proof that $B\ a_0 \sim_{\mathsf{U}_{\mathsf{eqv}}} B\ a_1$. Before proving quotient univalence this did not mean much, but *with* quotient univalence we now know how to interpret the relation on the universe and thus can push this further and conclude the much stronger assertion that the type $B\ a_0$ is equivalent to $B\ a_1$: Related terms in the domain of a type family are always sent to

equivalent types automatically equipped with functions that can transport terms back and forth between one another.

What does this mean for our example? To demonstrate, let us assume we also defined a second type of locally nameless lambda terms, lnLC. Locally nameless lambda terms are simply lambda terms where free variables are named, but all bound variables are referred to using De Bruijn indices. In the case of locally nameless lambda terms, alpha-equivalence is a non-issue as naming is determined uniquely by the structure of the term (as all bound variables are De Bruijn), and thus the type lnLC is equipped with the trivial reflexive relation. Let us assume we have defined the equivalence $(f, g) : \mathsf{LC} \simeq \mathsf{lnLC}$.

We will now use quotient univalence to transport a proof of progress of the untyped lambda calculus defined over the type of locally nameless lambda terms to type of named lambda terms. Why might one want to do this? In this case the proof is rather easy to define directly over named lambda terms, but often when proving properties of more expressive and realistic languages it is desirable to not complicate proofs with alpha-equivalence related issues and work with De Bruijn indices, but one also wants the final theorem statement to be about the named version of the language as it is the closest mathematical representation to the actual programming language. For this to work, let us assume we have defined a type a type of lambda algebras LAlg, which can equivalently be thought as the signature of a module containing the abstract interface used for defining lambda terms, and defined two terms of type LAlg, alg and lnalg, that instantiate the signature for both LC and lnLC respectively. The relation given by quotient univalence between LC and lnLC then lifts to define a witness relating the terms alg and lnalg. Let us also assume we have defined an abstract notion of the step relation for lambda algebras along with a predicate $P : \mathsf{LAlg} \to \mathsf{U}_{\mathsf{eqv}}$ that states the property of progress: every lambda term can either step or is a variable or a function. We can use the fact we automatically know the function P preserves relatedness to conclude that $P\ \mathsf{alg}$ is related to $P\ \mathsf{lnalg}$, and then use quotient univalence to extract the corresponding equivalence which in particular contains a function with type $P\ \mathsf{lnalg} \to P\ \mathsf{alg}$: Given a proof

p of type P lnalg stating that the locally nameless lambda terms satisfy progress, we can automatically generate the corresponding proof of type P alg concluding the same property for the named lambda terms.

The key takeaway of this example is that, by adding more structure to the type theory in an intelligent and organized manner, we did additional labor as the researchers when defining the theory that ultimately gives the user of it access to more true statements about their terms and types for free. I hope that in working through this light-hearted example the pattern of research in type theory I personally use as a fundamental framework will now be more visible and intuitive to you, particularly while reading this thesis.

A Note on Actual Quotient Type Theories

While my example of quotient type theory in this section is not serious, there are a few rigorous type theories that approach this problem that are quickly worth mentioning. First, extensional type theory is type theory with equality reflection, meaning any propositional equality can be internalized into a judgmental one [51]; this logically consistent approach solves the challenge of maneuvering around propositional equality proofs at the cost of decidable type checking. That being said, the NuPRL proof assistant is a working implementation of computational type theory that satisfies equality reflection resulting in a type theory with many extensional properties [25], and has been followed by a number of spiritual successors [8, 38]. On the intensional side of things, and the closest to the fake theory described in this section, Martin Hofmann's thesis *Extensional Concepts in Intensional Type Theory* explores an actual approach to setoid type theory [40]. Most recently, XTT is a type theory with uniqueness of identity proofs for its propositional equality that is based on Cartesian cubical type theory [72].

1.3 Technical Contributions

Directed type theory presents a number of unique challenges beyond those found in homotopy type theory, and as such a number of techniques have been explored to account for these nuances. Most notably, in directed type theory a universe of covariant type families will not contain all of the usual types of type theory—e.g., Π is contravariant, not covariant, in its domain. One approach that has been taken by directed type theory is to employ some kind of modal typing discipline, tracking variances or preventing certain uses of variables [45,52,54], so that being a covariant family is part of the typing of a term. Another approach, recently developed by Riehl and Shulman [62], is based on instead equipping each type with *both* a notion of path and a separate notion of directed morphism (representing the arrows in a category). In this approach, all existing homotopy type theory can be interpreted using the path structure on each type, providing a rich language of possible constructions. Being a covariant family is an *internally definable property* of a type, represented by another type that can be inhabited for a variety of type constructors—i.e. covariance is a type class.

While ordinary homotopy type theory has models in *simplicial sets*, the Riehl-Shulman type theory is based on a model in *bisimplicial sets*, $\mathbf{Set}^{\Delta^{\mathrm{op}}\times\Delta^{\mathrm{op}}}$, where the two copies of the simplex category Δ represent these two notions of path and directed morphism. While this suffices for formalizing mathematics, for applications to computer science we would like a computational interpretation of the type theory, and a constructive account of even ordinary homotopy type theory in simplicial sets has not fully been worked out (see [34] for some progress in this direction).

In this thesis, in place of a bisimplicial model of directed type theory, we consider a *bicubical* one, so that we can exploit the techniques that have been used to give constructive interpretations of ordinary homotopy type theory. We give a model of directed type theory with directed univalence in bicubical sets. This model is constructive in the sense of being defined in a constructive metatheory; we leave a more explicit operational semantics to

future work. We have formalized[2] much of the model in Agda using the internal language approach to presheaf models of homotopy type theory [14, 47, 57, 58]. In this approach, an extensional type theory (or Agda with some axioms as a substitute [40]) is used to describe cubical-set level constructions, and the main task is to program the definitions/proofs of the *Kan operations*, which describe how to transport in each type family. Bicubical sets is also a presheaf topos, and thus we too can leverage this technique for our work on bicubical directed type theory.

Chapters 3 and 4 focus on defining bicubical type theory and constructing directed univalence. First, Section 3.1 describes the axioms we add to the Agda proof assistant to work in its internal language and justifies their soundness with respect to the category of bicubical sets. In Sections 3.2 and 3.3, we use this internal language to define numerous classes of types including covariant discrete fibrations (covariant functors valued in types that themselves have trivial morphism structure) and universes containing the types of a couple of these classes (including one for the covariant discrete fibrations). In Section 4.1, we show that functions are a *reflection* of morphisms in this universe: every morphism determines a function, and vice versa; converting a function to a morphism and back is the identity; but for the other round-trip there seems to be only a morphism in one direction, not a path. This reflection is most of the directed univalence *equivalence* between functions and morphisms, but to complete the equivalence, it is necessary to invert this morphism, yielding a path. To do this, Cavallo, Riehl and Sattler's proof of directed univalence in the bisimplicial model [21, 68] uses a fact about the Reedy model structure on bisimplicial sets, that weak equivalences (and therefore homotopy equivalences between bifibrant objects) are *objectwise*—this means that if a morphism is an equivalence for each object of the directed category separately, then it is an equivalence. For our constructive model, we use a cubical, not simplicial, directed category (specifically, the "Dedekind" cube category with faces, degeneracies, symmetries, diagonals, and connections, but no reversals) so that we can build

[2] See the `agda/directed` directory of https://github.com/mzweav/phd, starting with `Thesis.agda` and `Summary.agda`.

universes of covariant fibrations constructively [47]. This cube category is not Reedy [67], so the proof for bisimplicial sets does not immediately apply to bicubical sets. However, for bisimplicial sets, the Reedy model structure coincides with the injective model structure (see e.g. [63, Example 7.8]), and the injective perspective applies to bicubical sets, allowing us to port this aspect of Cavallo, Riehl and Sattler's proof. Recently, Shulman [71] gave a new characterization of the fibrations in the injective model structure as objectwise fibrations equipped with an algebra for the *cobar construction* (see e.g. [60, 70]). Coquand, Sattler and Ruch [26, 28] give a constructive analogue of this definition as a special case of sheaf models (though the precise connection with the injective model structure remains to be worked out). In Section 4.2, we describe an Agda formalization of this approach, and check that the axioms describing this extension are true in bicubical sets. We then add an axiom covEquivAx that is true for this refined notion of fibration, and use it to complete the directed univalence equivalence for a refined covariant universe. We can summarize the key results contained in Chapters 3 and 4 using the framework from Section 1.2.

Semantic Structure	bicubical sets (Section 3.1)
Structural Syntax	interval types, cofibration logic, extension and glue types (Section 3.1)
The Properties	covariant composition, Segal types and cobar modal types (Sections 3.2 and 4.2)
The Universes	the universes of modal Segal types and modal covariant types (Sections 3.3, 3.4 and 4.2.3)
The Universal Property	constructive directed univalence (Sections 4.1 and 4.2.4)

Chapter 5 describes a generalization of the fibrant Riehl-Shulman extension types defined in [62], and then leverages this notion to define a type theory in which users can freely define constructive LOPS universes [47] within the syntax. In Section 5.1, we show that any variables of types corresponding to representables in the presheaf model that are quantified by a Π-type can freely be used in boundary conditions of extension types and as a part of cofibration witness types while preserving the fibrancy of the resulting type; furthermore, one can also quantify over variable cofibrations freely and use the variable cofibrations as well in any extension type boundary or cofibration witness type. This is made possible by

opening up our reasoning about fibrancy to consider when types are fibrant with respect to only a portion of the context. We prove this in the setting of an arbitrary presheaf category with respect to a new notion of fibrancy that is given by a generic filling problem. This abstraction is general enough to capture all of the current presheaf models of type theories and their classifications of types specified by filling problems. Having defined this new theory of "fiberwise, contextual fibrancy," we then define a syntax for a type theory that includes a constructor for LOPS universes in Section 5.2.3. Not only does this constructor allow the user to define and inhabit new universes classifying types for any notion of fibrancy given by the generic filling problem from within the syntax, but these universes can even be nested within one another, i.e. the user can carve a new LOPS universe out of an existing one, resulting in the universe of types from the first universe that additionally satisfy the fibrancy condition used to define the new universe. Considering an implementation that incorporates this universe constructor, the solutions to the filling problems that accompany each type former can be provided internally as library code and no longer must be manually incorporated as part of the implementation; this would substantially decreases the effort for implementing proof assistants based on types theories of this form, while allowing for more flexibility in the filling algorithms. This also makes it possible for users to provide codes for fibrant types that could not be covered by the generic cases hard-coded into the implementation when using the approach where the LOPS construction is not exposed.

Lastly, in Chapter 6 we overview our implementation of bicubical directed type theory: directed cooltt [3]. While modeled by the theory described in Chapters 3 and 4, the implementation contains a few interesting design decisions in regards to its syntax. Most notably, as opposed to exposing the connections for the directed interval we chose to instead include inequality cofibrations directly in the syntax and omit connections, resulting in a syntax that is sufficiently expressive for any use case we have considered while resulting in an implementation with a substantially more computationally efficient cofibration solver.

1.4 Related Work

Our proof of the directed univalence reflection does not overlap very much with Cavallo, Riehl and Sattler's directed univalence for bisimplicial sets [21, 68]—for example, we use the "glue" types used to define univalence in constructive cubical models [24], while their proof uses a dual type (called "weld" types in [55]), and we use the LOPS construction [47] to build universes constructively; however, for completing the reflection to an equivalence, we do use a lemma from their proof, which follows from objectwise homotopy equivalences inducing homotopy equivalences.

Buchholtz and Weinberger [17, 18] give some extensions of the Riehl-Shulman directed type theory, such as adding opposite categories, defining cartesian fibrations, and defining a universe of simplicial spaces inside of cubical spaces via a sheaf condition. In future work, we plan to investigate whether the techniques we develop here can be used to give a constructive account of directed univalence for a universe of cartesian fibrations (categories and functors, instead of groupoids and functors) as well.

North uses a distinct approach to define a type theory for ∞-categories in which the morphism types interpret directly into **Cat** [53]. The focus of North's work is to define a directed type theory compatible with expressing not only ∞-categories, but also directed spaces. This shift in priority results in a number of different design decisions, but the biggest from the perspective of this thesis is that—as with the other definitions of directed type theory (besides that described in this thesis)—it is not constructive.

Bicubical sets have also been used for type theory with parametricity: in Nuyts' approach to parametricity for (non-homotopy) type theory [55], both cube categories are Cartesian, and there is an inclusion from one interval the other. Here, we do not take an inclusion from the directed interval to the interval to be part of the cube category (but paths can be turned into morphisms in Kan types). In Cavallo and Harper's approach to parametricity for homotopy type theory [20], the cube category used for paths is Cartesian, while the cube category used for relations is semi-Cartesian (affine logic). In contrast, our directed cube

category also supports diagonals and connections to encode the simplicial shapes used by Riehl and Shulman [62].

In his thesis, Nuyts explores generalized notions of fibrancy that covers a notion of contextual fibrancy similar to that described in Chapter 5 [56, Chapter 8]. His definition of fibrancy is much more abstract and thus captures a much larger class of concepts than that which we consider, with contextual fibrancy simply being an example. By focusing specifically on contextual fibrancy, we develop much more specialized and powerful results for this specific case. In addition, we differ by representing these ideas mostly within the internal logic of presheaf categories, while Nuyts' work is defined externally using the language of weak factorization systems.

Additionally, a group lead by Riehl began some initial investigations into the notion of fibrancy versus fiber-wise (or contextual) fibrancy in the setting of bisimplicial directed type theory at the 2017 Mathematics Research Community at the Snowbird Resort in Utah.

Recently, Kudasov implemented Rzk: A proof assistant based upon Riehl and Shulman's bisimplicial directed type theory [4]. Rzk has proven to be a wonderful setting in which to formalize ∞-category theory synthetically, being used for ongoing formalizations of simplicial homotopy type theory [5, 6]. The primary downside to Rzk that is rectified by our implementation is that it is based on the bisimplicial model of directed type theory, and thus is inherently nonconstructive; while Rzk is consistent as a logic, it contains terms that cannot compute and thus the proof assistant cannot double as a programming language with normalization or canonicity. By basing directed cooltt on our bicubical model, our proof assistant is built on top of a constructive logic that ultimately can be equipped with a normalization/canonicity proof and a compiler, opening the potential use-space to include both formalizing mathematics (as Rzk supports) *and* applications in computer science, such as software verification.

Chapter 2

Background

This thesis builds upon a gargantuan mountain of prior work from numerous theories of mathematics, the scale of which makes painting a sufficiently detailed image a task incapable of fitting within a thesis of reasonable length. As such, I decided on an approach of picking and choosing the most central foundational themes of my thesis, and go into detail in those places I find the readily available literature (or more accurately the references I personally have encountered throughout my studies) lacking or outdated.

As a result of this decision, this chapter is structured as follows. First, in Section 2.1, I simply cover the bare minimum of the fundamentals of homotopy type theory; for a more comprehensive introduction to the field, I would direct you to the canonical reference, The HoTT Book [74], along with Rijke's textbook [64]. The mosts comprehensive section is Section 2.2, containing a dense but relatively complete introduction to categorical logic, incrementally building up the knowledge required to understand the inner workings of the internal language of a topos and its connection to dependent type theory. Finally, Section 2.3 introduces the basics of cubical type theory and Section 2.4 describes the prior work on directed type theory, the combination of which gave birth to the research explored in the remainder of my thesis.

2.1 Homotopy Type Theory

An important observation was made at the end of the 20th century: Intuitionistic type theory—while always considered until that point as a logic for reasoning over sets—was fully consistent with more general interpretations. Historically, the intended model of intuitionistic type theory was extensional, meaning that all proofs of equality are themselves always equal; this new direction of work broke away from this idea, resulting in not only new models of intuitionistic type theory, but entirely new type theories that fundamentally take advantage of this newfound flexibility surrounding the meaning of equality. In the 1990's the rules of intuitionistic type theory were found fully consistent when interpreting types as setoids, and then shown to also be consistent when interpreting types as groupoids; furthermore, the universe of this groupoid type theory exhibits what today we would call a univalence principle [40, 42]. In the following decade this idea expanded to cover ∞-groupoids, and with this jump in dimension the notion of homotopy levels of types was introduced along with the univalence axiom [11, 76, 77]. At this point, research in higher-dimensional type theory became an increasingly studied topic [33, 35, 36, 46, 48, 75, 78], and somewhere along the way this body of work created the field of homotopy type theory: the study of extensions of Martin-Löf type theory where types no longer represent sets or even groupoids; types can model topological spaces.

Given homotopy type theory is a rather extensive topic, and the HoTT Book [74] and Rijke's textbook [64] already provides an accessible and rigorous introduction to the basic ideas of homotopy type theory, I will not attempt to force an abridged summary into the span of a few pages in this section; instead, the remainder of this section works as an index, containing a few of the most fundamental facts, definitions and notations from homotopy type theory used throughout this thesis.

2.1.1 Key Facts and Definitions

Definition 2.1.1. The *homotopy fiber*, or *hfiber*, of a function $f : A \to B$ at a point $b : B$ is the type classifying the inverse image of b up to homotopy.

$$\mathsf{HFiber}(f, b) := \Sigma a : A.\mathsf{Path}_B\,(f\ a,\ b)$$

Definition 2.1.2. A type A is *contractible* if one can provide a term $a_0 : A$ and, for any other term $a : A$, a path from a_0 to a.

$$\mathsf{iscontr}\ A := \Sigma a_0 : A.\Pi a : A.\mathsf{Path}_A\,(a_0,\ a)$$

Definition 2.1.3. Given types A and B, a function $f : A \to B$ is an *equivalence* if there exists a $g : B \to A$ such that f and g are inverses up to homotopy:

$$\mathsf{isEquiv}\ f := \Sigma g : B \to A.\mathsf{Path}_{A\to A}\,(g \circ f, \mathsf{Id}\ A) \times \mathsf{Path}_{B\to B}\,(f \circ g, \mathsf{Id}\ B)$$

Two types A and B are *equivalent* if there exists an equivalence $f : A \to B$.

$$\mathsf{Equiv}\ A\ B := \Sigma f : A \to B.\mathsf{isEquiv}\ f$$

Definition 2.1.4 (Univalence)**.** A universe U is *univalent* if the type theory contains a term of the following type.

$$\Pi A, B : \mathsf{U}.\mathsf{Equiv}\ (\mathsf{Path}_U\,(A,\ B))\ (\mathsf{Equiv}\ A\ B)$$

2.2 Categorical Logic

Just as there are many different ways to define a topos (a mathematical structure we shall incrementally define over the course of this section), the question of "What is a type theory?" has many answers. While many are correct, the perspective most relevant to this thesis is that (dependent) type theory is the internal language of a topos. Conversely, the most useful interpretation of a topos for reading this thesis is that it is a category into which one can faithfully interpret dependent type theory. In this section, we will build up our understanding of categorical logic to ultimately explore the connection between topoi and type theory, focusing on the foundation needed for the techniques we utilize throughout the thesis.

For this section, we assume the reader is proficient with the basic definitions of category theory (e.g. category, functor, natural transformation), and ideally has some exposure to the concepts of limits and adjunctions. Should one wish to freshen up one's background before jumping into this tutorial on categorical logic—or if one desires a reference to aid their read through—we recommend Riehl's book *Category Theory in Context* [61];[1] the first four chapters of the text provide more than enough of a foundation to comfortably jump into the basics of categorical logic as we describe within the following pages.

To begin with, let us first view one (likely incomprehensible) definition of a topos.

Definition 2.2.1. A *topos* is a Cartesian closed category that has all finite limits and contains a subobject classifier.

Even ignoring the fact it builds upon a number of other definitions we have yet to define, there clearly is a large, nontrivial leap between this succinct definition of a topos and the realization that any topos provides sufficient structure into which one can always interpret dependent type theory. Let us work through the reasoning together and develop

[1] The text is freely (and legally) available online for personal use at https://emilyriehl.github.io/files/context.pdf.

our understanding of one of the many observations that makes topoi such a fruitful class of categories.

To start with, we will first explore the very basic ideas of categorical logic in Section 2.2.1. We then will shift our focus to the categorical logic of type theory by uncovering how every Cartesian closed category provides a model of the simply types lambda calculus in Section 2.2.2. Finally, in Section 2.2.3 we will conclude this introduction of categorical logic by first extending our model to locally Cartesian closed categories to capture the behavior of dependent type theory, and then defining topoi and modeling extensional dependent type theory in their internal language.

2.2.1 Syntactic Categories and Internal Languages

In order to discuss the connections between syntax and category theory in a mathematically sound way, we first must fix a formal definition of the notion of syntax.

Definition 2.2.2. For the purposes of this section, a *syntax* for a type theory is given by a set of contexts Σ and for every pair of contexts $\Delta, \Gamma \in \Sigma$ a set of substitutions $\Delta \vdash \Gamma$ closed under identity and composition. Composition of substitutions must be associative, as must the identity substitution be both a left and right unit to composition. We often will refer to the syntax by just the set Σ of contexts when it is clear to do so. To denote γ as a substitution in $\Delta \vdash \Gamma$, we write $\Delta \vdash \gamma : \Gamma$.

This particular notion of a type theory reduces its structure to a singular typing judgment for context substitutions; as most type theories contain a product type there generally is no meaningful distinction between the contexts and the types (at most differing by a strict isomorphism), and thus the judgment is equivalently the typing judgment for terms. Another important note to make is that the contexts are not limited to being simply a list or telescope of types (as is the case in this thesis, where we instead have multiple lists/telescopes for different classes of types), and even if contexts are given as lists of types they need not

represent the standard product of the types but instead can be some monoidal product (as is required for e.g. linear type theory).

Definition 2.2.3. Given a syntax Σ, the *syntactic category* has as objects the contexts, and for $\Delta, \Gamma \in \Sigma$, the morphisms from Δ to Γ are given by the set of substitutions $\Delta \vdash \Gamma$.

Clearly, the syntactic category is always a category, as the definition of a syntax has the category axioms built in (as any reasonable notion of substitution should indeed include identity substitutions, should compose, and the order of computing compositions of substitutions should not have any effect on the resulting substitution). The benefit of thinking of our syntax as a category is that it opens up the potential to identify abstract categorical structures that appear within the rules that define the type theory, and thus allows us to leverage existing mathematics about the categorical structures to conclude facts about the syntax and its properties more easily. The intuition that comes from this perspective also flows in the other direction, as often categorical constructions are helpful to inspire new syntax that will inherit the behavior and properties known from the mathematics. One notable example of this pathway from mathematics to syntax is the creation of homotopy type theory and all of the theories that have been built upon its foundation (and therefore includes this thesis).

Definition 2.2.4. Given a category $\mathcal{C}$, its *internal language* is a syntax with contexts being the objects in $\mathcal{C}$, and for $\Delta, \Gamma \in \mathcal{C}$ the substitutions $\Delta \vdash \Gamma$ are given by the the morphisms from Δ to Γ in $\mathcal{C}$.

To cover our bases, first observe that the above definition does yield a syntax, as the properties required of the substitution sets in the definition of a syntax match those imposed on the hom-sets of a category.

This formal definition, while correct, is in many ways too general and misses the more subtle meaning often associated with the term "internal language" in the same way the formal definition of syntax also leaves out many of the details we associate with the usage

of the word. The high level idea is this: The internal language of a category is a type theory where contexts/types are given by the objects of the category and terms are given by the morphisms; moreover, and what the generic definition above omits, we often include a syntactic system of inference rules describing (a subtheory of) the internal language that is generated directly from the various universal constructs contained within the category. This selection of inference rules is generally not unique, fixed, nor complete for a given category (although there are standard choices for the most common classes of categories used in categorical logic, as we will see in the following few sections). That being said, it is this notion of the internal language that is the most important conceptually as it truly describes a fully syntactic representation of the logic contained within the structure of the category, providing a clear, consistent interface between a syntactic logic and the category that describes the logic. We will quickly make this idea concrete in Section 2.2.2.

2.2.2 Cartesian Closed Categories and STLC

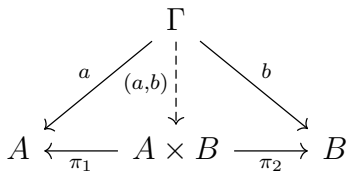
The first milestone on our journey towards understanding how topoi model dependent type theory will be to uncover how the internal language of any Cartesian closed category contains the simply typed lambda calculus. In order to do that, we begin by exploring categories with binary products.

The Internal Language of A Category with Binary Products

As we mentioned in the previous section, the internal language in practice can be represented by a system of syntactic inference rules corresponding to universal constructions in the category; but what do we mean by universal constructions? While it is only one class of such things, a common and intuitive example of universal constructions is limits. A limit defines an object determined from some data that is in particular paired with a unique structural way to map into the object along with morphisms describing how to project out the data used in its definition. These three pieces neatly align with syntactic counterparts: a formation rule, an

introduction rule, and some elimination rules; in addition, the equations that are given with the limit give the standard β-reduction rules, and the uniqueness property accompanying how to define morphisms into the object allow one to always derive an η-expansion rule.

To make this idea concrete, let us consider the canonical example of a limit: the binary product. Assume our category $\mathcal{C}$ is closed with respect to binary products. Then, for any two objects A and B in $\mathcal{C}$, there is an object $A \times B$ (the formation rule) with two projection maps $\pi_1 \in \mathcal{C}(A \times B, A)$ and $\pi_2 \in \mathcal{C}(A \times B, B)$ (the elimination rules). Lastly, for any $\Gamma \in \mathcal{C}$ and pair of maps $a \in \mathcal{C}(\Gamma, A)$ and $b \in \mathcal{C}(\Gamma, B)$ there is a unique map $(a, b) \in \mathcal{C}(\Gamma, A \times B)$ such that (a, b) followed by π_1 equals a and (a, b) followed by π_2 equals b (the introduction rule). We summarize the product with the commuting diagram below:



Conveniently, these rules naturally can be represented as inference rules in the internal language. We will write $A \in \mathsf{Type}$ to denote the judgment that A is a type in the syntax.

$$\frac{A \in \mathsf{Type} \qquad B \in \mathsf{Type}}{A \times B \in \mathsf{Type}} \qquad \frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B}$$

$$\frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \pi_1\ p : A} \qquad \frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \pi_2\ p : B}$$

To help our syntactic representation match our intuition, we write $f\ g$ to denote the morphism $f \circ g$ (as e.g. $\pi_1\ p$ in the above rule represents the morphism $\pi_1 \circ p \in \mathcal{C}(\Gamma, A)$).

An important additional step is to now consider the equations that hold for products in the internal language. In particular, given the rules thus far look like the standard definition of products in type theory, we also anticipate that the β-rules should hold.

$$\frac{\Gamma \vdash (a, b) : A \times B}{\pi_1(a, b) = a} \qquad \frac{\Gamma \vdash (a, b) : A \times B}{\pi_2(a, b) = b}$$

Conveniently, the commuting diagram above describing the product precisely expresses the two equations for the β-rules. Furthermore, as $\pi_1 \circ (\pi_1, \pi_2) = \pi_1 = \pi_1 \circ \mathrm{id}_{A \times B}$ and $\pi_2 \circ (\pi_1, \pi_2) = \pi_2 = \pi_2 \circ \mathrm{id}_{A \times B}$ we use uniqueness to conclude that (π_1, π_2) equals the identity map on $A \times B$. When represented syntactically in the internal language, this is simply the η-rule:

$$\frac{\Gamma \vdash p : A \times B}{(\pi_1\ p, \pi_2\ p) = p}$$

Thus, the internal language of a category with binary products supports the standard inference rules of binary product types.

As the key takeaway of this example, we typically think of the internal language of a category with binary products as not simply being the opaque typing judgment from the formal definition of "internal language" given in Definition 2.2.4, but the formal judgment paired with the above syntactic inference rules.

The Internal Language of a Cartesian Category

Definition 2.2.5. A *Cartesian category* is a category closed under all finite products. Note this includes the nullary product 1, yielding the terminal object. Equivalently, a Cartesian category is a category containing a terminal object that is closed under binary products.

Let us now consider the internal language of a Cartesian category. For starters, we will include every rule pertaining to products, as indeed a Cartesian closed category contains all binary products. Additionally, as it contains a terminal object 1, we add the standard rules for the unit type:

$$\frac{}{1 \in \mathsf{Type}} \qquad \frac{\Gamma \vdash}{\Gamma \vdash * : 1} \qquad \frac{\Gamma \vdash x : 1}{x = *}$$

The unit type is a type (as it is an object in the category); for any context Γ there is a term $*$ in the unit type (as for any object there is a morphism to the terminal object); and any term of the unit type is equal to the canonical term $*$ (as the terminal morphism is always unique).

As potentially a more important consequence than unit and product types themselves, the combination of binary products and a terminal object allows one to define the typical structural rules for (non-dependent) contexts. While contexts and types alway coincide in the formal notion of internal language as defined in Definition 2.2.4, there is more flexibility in the syntactic inference rules (as deciding the selection of rules involves the subjective choice of notation) and thus we will add a new syntactic judgment $\Gamma \vdash$ to indicate Γ is a context in the syntax. Using just the basic structure of products and the terminal object, we can derive the following rules for contexts and variables.

$$\frac{}{\cdot \vdash} \qquad \frac{\Gamma \vdash \qquad A \in \mathsf{Type}}{\Gamma, A \vdash}$$

$$\frac{\Gamma, A \vdash}{\Gamma, A \vdash \mathsf{var} : A} \qquad \frac{\Gamma \vdash b : B \qquad A \in \mathsf{Type}}{\Gamma, A \vdash \mathsf{wk}_A\ b : B}$$

1. There is an empty context (the terminal object).

2. Given a context and a type we can extend the context with said type; we write Γ, A as an alternative syntax for $\Gamma \times A$.

3. We can project out the outermost variable from the context.

4. Given a term in a context we can always weaken the context and still have the analogous term of the same type (with an explicit substitution applied to the term, as the internal language effectively uses DeBruijn indices given how we defined context extension).

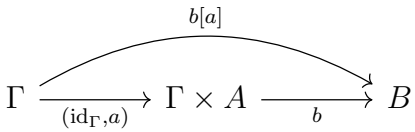
To help with syntactic intuition, we write var to denote π_2 when used to project the outermost variable of the context, and $\mathsf{wk}_A\ b$ to denote $b \circ \pi_1$ when weakening a term b by adding new variable of type A to the context.

Contexts exists not only to provide scope for open terms in a theory, but in particular to indicate the structure of substitutions for open variables; as such, we also should anticipate the internal language to be able to express variable substitutions now that contexts are

structured. Let us first write down and explain the rule we will add to the internal language before justifying it categorically.

$$\frac{\Gamma, A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash b[a] : B}$$

As the internal language effectively uses DeBruijn indices and thus has no notion of variables, we write $b[a]$ to express that a is being substituted into b for the outermost variable. We now define the morphism $b[a] \in \mathcal{C}(\Gamma, B)$ as $b \circ (\mathrm{id}_\Gamma, a)$.

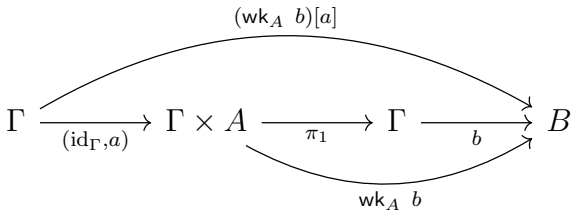


Lastly, having defined weakening and substitution, we would hope to be able to conclude that if we weaken a term and then apply a substitution to replace the variable abstracted by weakening, we then recover the original term.

Lemma 2.2.6. *The following inference rule holds in the internal language of a Cartesian category.*

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B}{(\mathsf{wk}_A\ b)[a] = b}$$

Proof. Unfolding our definitions, we depict the morphism $(\mathsf{wk}_A\ b)[a]$ as shown here.



As $\pi_1 \circ (\mathrm{id}_\Gamma, a) = \mathrm{id}_\Gamma$, it follows that indeed $(\mathsf{wk}_A\ b)[a] = b$. ⊠

The Internal Language of a Cartesian Closed Category

While the "Cartesian" moniker afforded us structural contexts, we now will see how the addition of "closed" contributes functions to the internal language of Cartesian closed categories.

To help us understand the definition of a Cartesian closed category, let us first define the concept of an adjunction.

Definition 2.2.7. Given two categories $\mathcal{C}$ and $\mathcal{D}$ and two functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{D} \to \mathcal{C}$, the pair of functors F and G form an *adjunction* between $\mathcal{C}$ and $\mathcal{D}$ if for every object $C \in \mathcal{C}$ and every $D \in \mathcal{D}$ there is a bijection of homsets $\mathcal{C}(C, G\ D) \simeq \mathcal{D}(F\ C, D)$. We call F the *left adjoint* and G the *right adjoint*, and we denote that F and G are an *adjoint pair* with the notation $F \dashv G$.

Definition 2.2.8. A *Cartesian closed category* is a Cartesian category $\mathcal{C}$ paired with a functor $[-,-] : \mathcal{C}^{\text{op}} \times \mathcal{C} \to \mathcal{C}$ such that for every object $A \in \mathcal{C}$, $- \times A \dashv [A, -]$ form an adjoint pair. Furthermore, the isomorphism from the adjunction $\lambda : \mathcal{C}(\Gamma \times A, B) \simeq \mathcal{C}(\Gamma, [A, B])$ is natural in Γ, A and B. We call the functor $[-,-]$ the *internal hom*. Notationally, we will often write B^A to denote the object $[A, B]$.

To provide some quick intuition, we can use the adjunction to quickly justify the name of the internal hom functor:

$$\mathcal{C}(A, B) \simeq \mathcal{C}(1 \times A, B) \simeq \mathcal{C}(1, B^A)$$

There is a natural isomorphism between the elements of the hom-set $\mathcal{C}(A, B)$ and the global elements of the object B^A.[2]

As expected, the internal language contains all of that described for Cartesian categories. Let us now consider the what is at this point the most interesting construction to incorporate into the internal language: the presence of the internal hom. The formation rule is

[2]For those unfamiliar, the global elements of an object A in a category $\mathcal{C}$ containing a terminal object 1 are given by the elements of the hom-set $\mathcal{C}(1, A)$. Conceptually, they are the categorical generalization of the set-theoretic notion of what it means to be an "element" of the object A.

straightforward, although note that we will change our syntax, using $A \to B$ in place of B^A.

$$\frac{A \in \mathsf{Type} \qquad B \in \mathsf{Type}}{A \to B \in \mathsf{Type}}$$

The introduction rule is also rather easy to translate. For any context Γ and types A and B (i.e. any three objects in the category), the adjunction of the internal hom provides the bijection $\lambda : \mathcal{C}(\Gamma \times A, B) \simeq \mathcal{C}(\Gamma, B^A)$. Rewriting the action of the bijection from left to right in the syntax of the internal language yields the following rule.

$$\frac{\Gamma, A \vdash b : B}{\Gamma \vdash \lambda b : A \to B}$$

Recalling our notational choice for Γ, A to be synonymous with the product, this is the standard introduction rule for function types: if b is a term of type B in a context extended by A, then there is a term λb binding A from the context with type $A \to B$.

Let us now work in the opposite direction, and write down the elimination rule we expect, and then justify it using the structure of the Cartesian closed category. The standard elimination rule for functions is shown below:

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f@a : B}$$

In order for this to hold in the internal language, we need to provide a construction that takes as input morphisms $f \in \mathcal{C}(\Gamma, B^A)$ and $a \in \mathcal{C}(\Gamma, A)$, and outputs the morphism $f@a \in \mathcal{C}(\Gamma, B)$. To do this, let us consider $\lambda^{-1} f \in \mathcal{C}(\Gamma \times A, B)$ taking f across the internal hom adjunction the opposite direction than that we used for the introduction rule. With this, we do get a morphism that lands in B, but we still need to fix the domain so it starts in Γ. As we have yet to use the morphism corresponding to the argument, it seems like a likely candidate to help complete our solution. Using the universal property of the product, we can define $(\mathrm{id}_\Gamma, a) \in \mathcal{C}(\Gamma, \Gamma \times A)$ (the same morphism used to define substitution for the outermost

variable in the context), and thus we can compose to construct a candidate morphism for application: $\lambda^{-1} f \circ (\mathrm{id}_\Gamma, a)$, or equivalently $(\lambda^{-1} f)[a]$.

$$\Gamma \xrightarrow[(\mathrm{id}_\Gamma, a)]{} \Gamma \times A \xrightarrow[\lambda^{-1} f]{} B \qquad \Gamma \xrightarrow{f@a} B$$

Just because the morphism proposed above "type checks" does not make it correct; we need to ensure it actually is the morphism corresponding to what we think of as function application. As an initial sanity check it does seem to do the right thing conceptually: First, λ^{-1} is unwrapping the body of the function, and then the morphism we precompose with is the substitution that swaps in a for the variable previously bound by the function. To verify this intuition, let us first derive the β-rule for function application.

Lemma 2.2.9. *The following inference rule holds for the internal language of any Cartesian closed category.*

$$\frac{\Gamma \vdash \lambda b : A \to B \qquad \Gamma \vdash a : A}{\lambda b @ a = b[a]}$$

Proof. Unfolding the definition, we can rewrite $\lambda b @ a$ as $(\lambda^{-1} \lambda b)[a]$, and as λ^{-1} is indeed the inverse of λ, we conclude that the β-rule holds. ⊠

To complete our derivation that functions are contained in the internal language, we must show it supports one last equation: η-expansion. In particular, we want to ensure the following.

Lemma 2.2.10. *In the internal language of any Cartesian closed category, the following inference rule holds true.*

$$\frac{\Gamma \vdash f : A \to B}{\lambda((\mathsf{wk}_A\ f) @ \mathsf{var}) = f}$$

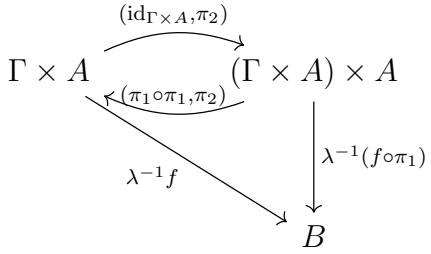
Proof. To begin, let us unfold the definition of $(\mathsf{wk}_A\ f) @ \mathsf{var}$.

$$\Gamma \times A \xrightarrow{(\mathrm{id}_{\Gamma \times A}, \pi_2)} \Gamma \times A \times A \xrightarrow{\lambda^{-1}(f \circ \pi_1)} B$$

We claim $\lambda(\lambda^{-1}(f \circ \pi_1) \circ (\mathrm{id}_{\Gamma \times A}, \pi_2)) = f$. First, let us take advantage of the bijection of the adjunction, and instead prove the equation after applying λ^{-1} to both sides: $\lambda^{-1}(f \circ \pi_1) \circ (\mathrm{id}_{\Gamma \times A}, \pi_2) = \lambda^{-1} f$. Now, recall that in the definition of a cartesian closed category, the bijection must be natural in all of the arguments, and in particular for this proof the zone representing the context (separate from the domain type). From this, we know the following square commutes in **Set**.

$$\begin{array}{ccc} \mathcal{C}(\Gamma, B^A) & \xrightarrow{\lambda^{-1}} & \mathcal{C}(\Gamma \times A, B) \\ {\scriptstyle -\circ\pi_1}\downarrow & & \downarrow{\scriptstyle -\circ(\pi_1, \mathrm{id}_A)} \\ \mathcal{C}(\Gamma \times A, B^A) & \xrightarrow[\lambda^{-1}]{} & \mathcal{C}((\Gamma \times A) \times A, B) \end{array}$$

Applying the corresponding equation to our morphism $f \in \mathcal{C}(\Gamma, B^A)$, we conclude $\lambda^{-1}(f \circ \pi_1) = \lambda^{-1} f \circ (\pi_1, \mathrm{id}_A)$. Finally, consider the diagram in $\mathcal{C}$ below.



First, note that the equation we derived from naturality above concludes that the diagram commutes from the top right (ignoring the morphism $(\mathrm{id}_{\Gamma \times A}, \pi_2)$). Second, the equation we wish to prove is that the diagram commutes from the top left (ignoring (π_1, id_A)). Precomposing the two equal morphisms beginning at the top right of the diagram by $(\mathrm{id}_{\Gamma \times A}, \pi_2)$ gives us that $\lambda^{-1}(f \circ \pi_1) \circ (\mathrm{id}_{\Gamma \times A}, \pi_2) = \lambda^{-1} f \circ (\pi_1, \mathrm{id}_A) \circ (\mathrm{id}_{\Gamma \times A}, \pi_2)$. Lastly, using the universal property of products, we can conclude that (π_1, id_A) is a retract of $(\mathrm{id}_{\Gamma \times A}, \pi_2)$, and thus the composition $(\pi_1, \mathrm{id}_A) \circ (\mathrm{id}_{\Gamma \times A}, \pi_2)$ is the identity map and $\lambda^{-1}(f \circ \pi_1) \circ (\mathrm{id}_{\Gamma \times A}, \pi_2) = \lambda^{-1} f$. Moving the equation through the bijection of the adjunction and rewriting with our notation, we have indeed proven the η-expansion rule for functions: $\lambda((\mathsf{wk}_A\ f)@\mathsf{var}) = f$. ⊠

The Type judgment:

$$\frac{}{1 \in \mathsf{Type}} \qquad \frac{A \in \mathsf{Type} \qquad B \in \mathsf{Type}}{A \times B \in \mathsf{Type}} \qquad \frac{A \in \mathsf{Type} \qquad B \in \mathsf{Type}}{A \to B \in \mathsf{Type}}$$

The Context judgment:

$$\frac{}{\cdot \vdash} \qquad \frac{\Gamma \vdash \qquad A \in \mathsf{Type}}{\Gamma, A \vdash}$$

The Term judgment:

$$\frac{\Gamma, A \vdash}{\Gamma, A \vdash \mathsf{var} : A} \qquad \frac{\Gamma \vdash b : B \qquad A \in \mathsf{Type}}{\Gamma, A \vdash \mathsf{wk}_A\ b : B}$$

$$\frac{\Gamma, A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash b[a] : B} \qquad \frac{\Gamma \vdash}{\Gamma \vdash * : 1}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B} \qquad \frac{\Gamma, A \vdash b : B}{\Gamma \vdash \lambda b : A \to B}$$

$$\frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \pi_1\ p : A} \qquad \frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \pi_2\ p : B} \qquad \frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f@a : B}$$

The Equality judgment:

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B}{(\mathsf{wk}_A\ b)[a] = b} \qquad \frac{\Gamma \vdash x : 1}{x = *}$$

$$\frac{\Gamma \vdash (a, b) : A \times B}{\pi_1\ (a, b) = a} \qquad \frac{\Gamma \vdash (a, b) : A \times B}{\pi_2\ (a, b) = b} \qquad \frac{\Gamma \vdash p : A \times B}{(\pi_1\ p, \pi_2\ p) = p}$$

$$\frac{\Gamma \vdash \lambda b : A \to B}{\lambda b@a = b[a]} \qquad \frac{\Gamma \vdash f : A \to B}{\lambda((\mathsf{wk}_A\ f)@\mathsf{var}) = f}$$

Figure 2.1: Inference rules for the simply typed lambda calculus

The Simply Typed Lambda Calculus

As we all know, the simply typed lambda calculus is the minimal core of purely typed functional programming languages, along with that of any syntactic type theory. We can describe the logic using four judgments: $A \in \mathsf{Type}$ stating that A is a type, $\Gamma \vdash$ stating that Γ is a context, $\Gamma \vdash a : A$ stating that a is a term with type A in context Γ, and $a_1 = a_2$ stating when two terms are equal. The inference rules in Figure 2.1 define the simply typed lambda calculus (with products).

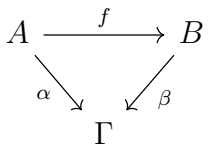
Not only does Figure 2.1 represent a relatively standard syntactic formulation of the simply typed lambda calculus, but we have shown that every single rule is consistent with the internal language of *any arbitrary* Cartesian closed category; furthermore, not that we will work through the construction here, the syntactic category of the simply typed lambda calculus is also a Cartesian closed category. As a result of these facts, we justify the canonical claim: "The simply typed lambda calculus is the internal language of Cartesian closed categories."

2.2.3 Topoi and Dependent Type Theory

Dependent Types in the Internal Language

Unsurprisingly, the key conceptual jump from the simply typed lambda calculus to dependent type theory is that types can depend on the variables in the context. Semantically this forces a shift in how we represent the types in category theory, as the objects in categorical logic represent contexts and now types in an open context are no longer contexts themselves (although the objects in our category can equivalently be thought of as closed types which do still coincide with contexts, as we can think of a dependent context as an iterated Σ-type); in order to model types from a category $\mathcal{C}$ depending on a context represented by $\Gamma \in \mathcal{C}$ we will actually shift our setting to the slice category $\mathcal{C}/\Gamma$.

As a quick refresher, recall that an object in the slice category $\mathcal{C}/\Gamma$ is given by an object $A \in \mathcal{C}$ paired with a morphism $\alpha \in \mathcal{C}(A, \Gamma)$, and a morphism $f \in \mathcal{C}/\Gamma((A, \alpha), (B, \beta))$ is a morphism $f \in \mathcal{C}(A, B)$ such that the following commutes.



A dependent type in context Γ is represented as an object $(A, \alpha) \in \mathcal{C}/\Gamma$. That being said, unlike with our previous work with the categorical semantics of simple types, the object A

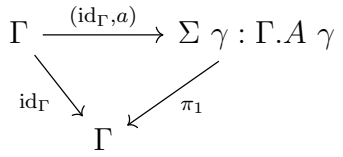
is no longer a direct representation of a syntactic type "$\gamma : \Gamma \vdash A\ \gamma \in \mathsf{Type}$," as the syntactic "$A$" is open, while the object A still represents a closed type. Instead, A represents what we call the total space of the dependent type, meaning the collection of all $A\ \gamma$ for every $\gamma : \Gamma$ all grouped together. The morphism α are the projection indicating how A depends on Γ. A key consequence of our category $\mathcal{C}$ ultimately being a topos is that it allows us to simplify this picture and work with something more familiar: Assuming $\mathcal{C}$ is a topos (or to be more precise, just a locally Cartesian closed category containing a terminal object), any object in the slice category (A, α) is isomorphic to the object $(\Sigma\ \gamma : \Gamma.\alpha^{-1}(\gamma), \pi_1)$, the pair consisting of 1. the dependent sum representing the closure of the dependent type family paired with its context and 2. the projection that returns the context. The inverse image of α at γ, written $\alpha^{-1}(\gamma)$ above, will be formally defined shortly as an instance of Definition 2.2.11. Given this isomorphism, without loss of generality (in most cases, which we will make precise shortly), we can think of the objects representing dependent types as always maintaining the form $\Sigma\ \gamma : \Gamma.A\ \gamma$ structurally, and simplify our notion of morphisms to those shown in the commutative diagram below where f can be thought of as a family of morphisms $f_\gamma \in \mathcal{C}(A\gamma, B\ \gamma)$.

$$\begin{array}{ccc} \Sigma\ \gamma : \Gamma.A\ \gamma & \xrightarrow{(\pi_1, f)} & \Sigma\ \gamma : \Gamma.B\ \gamma \\ & {\scriptstyle \pi_1} \searrow \quad \swarrow {\scriptstyle \pi_1} & \\ & \Gamma & \end{array}$$

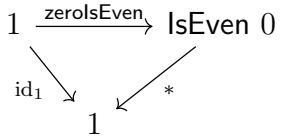
While knowing one can always refactor a dependent type into this form is certainly helpful conceptually—particularly when first approaching categorical logic assuming a preexisting background in syntactic type theory (and for this reason I will be maintaining this practice for the remainder of this section)—in practice one often does not refactor and instead works with arbitrary objects in the slice category and simply retains the conceptual idea of the dependent type as its total space and context projection.

Having shown how to represent dependent types within category theory, the natural progression is to now ask "how does one represent terms of dependent types?" Specifically, let us consider what the categorical analogue of the judgment $\gamma : \Gamma \vdash a : A\ \gamma$ will be. For

starters, we know that our dependent type will be represented as the object of $\mathcal{C}/\Gamma$ given by the dependent sum $\Sigma\ \gamma : \Gamma.A\ \gamma$ paired with its first projection, and in the spirit of categorical logic a should be a morphism into this object from that representing the context; but which object in $\mathcal{C}/\Gamma$ represents the context? It turns out the correct answer to this question is Γ paired with its identity morphism: $(\Gamma, \mathrm{id}_\Gamma)$. In particular, this means that terms a satisfying the judgment $\gamma : \Gamma \vdash a : A\ \gamma$ correspond to morphisms for which the diagram below commutes.

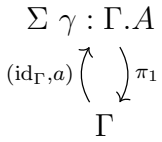


Analyzing this picture, we begin to see how using the slice category $\mathcal{C}/\Gamma$ captures the meaning of types depending on context Γ. As we had when working in Cartesian closed categories, the term a is a morphism out of the context Γ into the object representing our type. That being said, a morphism from Γ to the total space of our dependent type A would not be sufficient for the purposes of dependent type theory, as a term in context $\gamma : \Gamma$ not only must land in A, but specifically in the portion of A that corresponds to it depending on γ; for example, it should not be the case that $\mathsf{zeroIsEven}$ represents a witness to the judgment $x : \mathbb{N} \vdash \mathsf{zeroIsEven} : \mathsf{IsEven}\ x$ when $\mathsf{zeroIsEven}$ always lands in $\mathsf{IsEven}\ 0$, as 0 does not always equal x given x being in the context signifies quantifying over all potential values of type $\mathbb{N}$. As $\mathsf{IsEven}\ 0$ is a type scoped by the empty context, the syntactic judgment $\cdot \vdash \mathsf{zeroIsEven} : \mathsf{IsEven}\ 0$ corresponds to the triangle below.

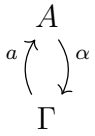


In general, to ensure terms agree with the dependency from context, terms must be a morphism into the total space of the dependent type such that it preserves its "location" in the context, encoded by the fact the triangle commutes.

As a quick remark on common practices, we can simplify the triangle corresponding to $\gamma : \Gamma \vdash a : A\ \gamma$ by collapsing the identity morphism to an equivalent statement: (id_Γ, a) is a section in the diagram below.



One often uses this representation to denote terms of dependent types instead of the triangle, given it is more compact. As a secondary comment, in situations where one cannot or prefers not to force the structure of the object representing the (dependent) type to that of a dependent sum, this picture is particularly concise. A term a of the type $(A, \alpha) \in \mathcal{C}/\Gamma$ corresponds to any section of α.



The last mechanization needed to model dependent types is to define the action of context substitutions on types and terms. We syntactically know the following rules.

$$\frac{\Gamma \vdash A \in \mathsf{Type} \qquad \Delta \vdash \sigma : \Gamma}{\Delta \vdash A[\sigma] \in \mathsf{Type}} \qquad \frac{\Gamma \vdash a : A \qquad \Delta \vdash \sigma : \Gamma}{\Delta \vdash a[\sigma] : A[\sigma]}$$

If A is a type in context Γ and σ is a substitution from Δ to Γ, then the substitution induces an action on types, supplying us $A[\sigma]$: a type in context Δ. Similarly, if a is a term of type A in context Γ, the substitution also induces a term $a[\sigma]$ of type $A[\sigma]$ in context Δ. As our dependent types are expressed as objects of a slice category and terms as morphisms of the slice category, both substitution actions correspond to needing a functor from $\mathcal{C}/\Gamma$ to $\mathcal{C}/\Delta$ for every morphism from Δ to Γ in $\mathcal{C}$ (for all object Δ and Γ). The functor that captures the correct behavior for context substitution (i.e matches our expectations from syntactic type theory) is called the base change functor.

Definition 2.2.11. Given a category $\mathcal{C}$ closed under pullbacks and a morphism $f \in \mathcal{C}(A, B)$, the *base change functor* $f^* : \mathcal{C}/B \to \mathcal{C}/A$ is defined by sending an object $x \in \mathcal{E}(X, B)$ to its pullback along f.

To illustrate the behavior of the base change functor, consider weakening a dependent type B in context Γ by a type A in context Γ. Specifically, this corresponds to applying the action of the projection $\mathsf{wk}_A : \Gamma, A \to \Gamma$ to B. The substitution action is depicted in the following pullback square.

$$\begin{array}{ccc} \mathsf{wk}_A^*\ (\Sigma\ \gamma : \Gamma.B\ \gamma) & \longrightarrow & \Sigma\ \gamma : \Gamma.B\ \gamma \\ {\scriptstyle \mathsf{wk}_A^*\ \pi_1}\downarrow & \lrcorner & \downarrow{\scriptstyle \pi_1} \\ \Gamma, A & \xrightarrow[\mathsf{wk}_A]{} & \Gamma \end{array}$$

Unfolding the definition of the pullback, we can easily workout the resulting type is (isomorphic to) $\Sigma\ (\gamma, a) : (\Gamma, A).B\ \gamma$, and the morphism into Γ, A is the projection π_1. Syntactically, this corresponds to the following inference.

$$\frac{\Gamma \vdash A \in \mathsf{Type} \qquad \Gamma \vdash B \in \mathsf{Type}}{\Gamma, A \vdash B[\mathsf{wk}_A] \in \mathsf{Type}}$$

Another important example of the action of substitution is that given by substituting in a value for the context. Consider a type B depending on a type A, along with a closed term $a : A$. The base change along the substitution yields the following square.

$$\begin{array}{ccc} B\ a & \xrightarrow{(a, \mathrm{id}_{B\ a})} & \Sigma x : A.B\ x \\ {\scriptstyle *}\downarrow & \lrcorner & \downarrow{\scriptstyle \pi_1} \\ 1 & \xrightarrow[a]{} & A \end{array}$$

In the case where we haven't already refactored our dependent type into the Σ closure of the context, the base change of an arbitrary object $(B, \beta) \in \mathcal{C}/A$ along $a \in \mathcal{C}(1, A)$ yields

the inverse image $\beta^{-1}(a)$, indeed being one of the fundamental constructions with which we define the refactorization itself. We also can generalize this to the base change along the substitution of a term a for the outermost variable of the context Γ, A as shown here.

$$\begin{array}{ccc} \Sigma\gamma : \Gamma.B\ \gamma\ a & \xhookrightarrow{(\pi_1, a, \pi_2 \circ \pi_2)} & \Sigma\gamma : \Gamma.\Sigma x : A\ \gamma.B\ \gamma\ x \\ \downarrow & \lrcorner & \downarrow \\ \Gamma & \xrightarrow[(\mathrm{id}_\Gamma, a)]{} & \Sigma\gamma : \Gamma.A\ \gamma \end{array}$$

Note that both vertical arrows are the projections to their codomain.

Quantifiers as Adjoints in Locally Cartesian Closed Categories

In order for a category to support dependent types, the property one needs is that the category is locally Cartesian closed.

Definition 2.2.12. A category $\mathcal{C}$ is *locally Cartesian closed* if for every object $A \in \mathcal{C}$ the slice category $\mathcal{C}/A$ is Cartesian closed.

What is arguably the most obvious consequence for the internal language of a locally Cartesian closed family is that the internal hom in the slice category provides a way to define function types between types that depend on the same context. In particular, the internal hom in the category $\mathcal{C}/\Gamma$ translates into the following inference rule (along with the expected rules for lambda introduction, function elimination, etc. . .).

$$\frac{\gamma : \Gamma \vdash A\ \gamma \in \mathsf{Type} \qquad \gamma : \Gamma \vdash B\ \gamma \in \mathsf{Type}}{\gamma : \Gamma \vdash A\ \gamma \to B\ \gamma \in \mathsf{Type}}$$

While certainly the above rule is useful and necessary for dependent type theory, expectations would include the more general rule for Π-types. Thinking back on the pattern for defining function types in the internal language of Cartesian closed categories, the crux of the categorical structure utilized is that the internal hom functor $[A, -]$ is the right adjoint

to the product functor $- \times A$, which we can equivalently think of as the context weakening functor. Might a similar pattern apply in the dependent setting? As the dependent analogue of weakening, we do already have the base change functor along the morphism $\mathsf{wk}_A \in \mathcal{C}(\Sigma\ \gamma : \Gamma . A\ \gamma, \Gamma)$ (i.e. the first projection): $\mathsf{wk}_A^* : \mathcal{C}/\Gamma \to \mathcal{C}/(\gamma : \Gamma, A\ \gamma)$. One might choose to investigate whether this functor happens to have a right adjoint itself, which we will denote by $\Pi_A : \mathcal{C}/(\gamma : \Gamma, A\ \gamma) \to \mathcal{C}/\Gamma$. Certainly the domain and codomain of this functor seem correct, as the formation rule for Π-types with domain A take as input a type depending on the context $\gamma : \Gamma, A\ \gamma$ and output a type binding A in context Γ. We also can surmise the adjunction would provide the structure needed to derive the correct behavior for terms of Π-types based upon our experience deriving the rules for function types in the STLC. Thankfully, it is the case that such an adjunction always exists for any locally Cartesian closed category; more generally, one can prove the following.

Theorem 2.2.13 ([32, §1.3], [69, §2.4])**.** *A category $\mathcal{C}$ with all pullbacks is locally Cartesian closed if and only if for every morphism f the base change functor f^* has a right adjoint Π_f.*

While the right adjoint exists for the base change functor for every morphism in $\mathcal{C}$, the Π-types we are accustomed to in syntax are always those arising from the base change along context weakening $\mathsf{wk}_A \in \mathcal{C}((\gamma : \Gamma, A\ \gamma), \Gamma)$. The adjunction yields the following natural bijection.

$$\lambda : \mathcal{C}/(\Gamma, A)(\mathsf{wk}_A^*\ X, B) \simeq \mathcal{C}/\Gamma(X, \Pi_A\ B)$$

If we instantiate the object X at $(\Gamma, \mathrm{id}_\Gamma) \in \mathcal{C}/\Gamma$, this bijection tells us that the terms of type $\Pi_A\ B$ in context Γ are precisely given by terms of type B in context Γ, A. Furthermore, given the machinery to represent dependency given by the slice categories, the type A can depend on the context Γ and B can depend on Γ and the type A. Following the same arguments as for function types in the internal language of Cartesian closed categories, we can conclude the inference rules below are consistent in the internal language of a locally Cartesian closed

category.

$$\frac{\gamma : \Gamma \vdash A\ \gamma \in \mathsf{Type} \qquad \gamma : \Gamma, x : A\ \gamma \vdash B\ \gamma\ x \in \mathsf{Type}}{\gamma : \Gamma \vdash \Pi x : A\ \gamma . B\ \gamma\ x \in \mathsf{Type}}$$

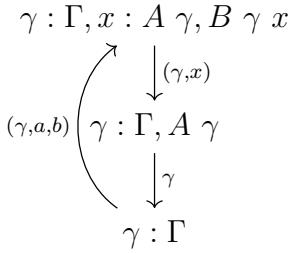
$$\frac{\gamma : \Gamma, x : A\ \gamma \vdash b : B\ \gamma\ x}{\gamma : \Gamma \vdash \lambda b : \Pi x : A\ \gamma . B\ \gamma\ x} \qquad \frac{\gamma : \Gamma \vdash f : \Pi x : A\ \gamma . B\ \gamma\ x \qquad \gamma : \Gamma \vdash a : A\ \gamma}{\gamma : \Gamma \vdash f@a : B\ \gamma\ a}$$

In the rule for function application, $f@a$ is given by $\lambda^{-1} f[a]$ where $[a]$ substitutes $x : A\ \gamma$ for a (itself given by the base change functor for the morphism $(\mathrm{id}_\Gamma, a) \in \mathcal{C}(\Gamma, \Sigma\gamma : \Gamma . A\ \gamma)$). The equations for β-reduction and η-expansion follow from the adjunction analogously to the derivations in Cartesian closed categories.

Having concluded that locally Cartesian closed categories support Π-types, let us now consider Σ-types. In the case where the locally Cartesian closed category contains a terminal object, the following theorem holds.

Theorem 2.2.14 ([32, §1.3], [69, §2.4])**.** *Consider a locally Cartesian closed category $\mathcal{C}$ containing a terminal object (and thus $\mathcal{C}$ is closed under all finite limits). For any morphism f, the base change functor f^* has a left adjoint Σ_f.*

If the notation didn't give it away, the left adjoint to base change ends up providing the categorical analogue of the Σ-type constructor. From the adjunction, we can identify that, for $f \in \mathcal{C}(\Delta, \Gamma)$ one way to compute the action of $\Sigma_f : \mathcal{C}/\Delta \to \mathcal{C}/\Gamma$ is simply by post composition: $\Sigma_f(A, \alpha) = (A, f \circ \alpha)$. As with Π-types, the rule is more general than needed syntactically, and in practice we only utilize the adjunction when instantiated at a context weakening morphism. Let us take a look at what happends when we consider terms of $\Sigma_A\ B$ in context Γ.



As we can see, by postcomposing with the weakening morphism we lose the dependency on $A\ \gamma$ in the slice category; as a result, the section corresponding to terms of type $\Sigma_A\ B$

in context Γ are pairs of any term $a : A$ in context Γ with a compatible $b : B\ a$ over Γ. This indeed aligns with the Σ-types found in syntactic type theory, with all of the expected inference rules.

$$\frac{\gamma : \Gamma \vdash A\ \gamma \in \mathsf{Type} \qquad \gamma : \Gamma, x : A\ \gamma \vdash B\ \gamma\ x \in \mathsf{Type}}{\gamma : \Gamma \vdash \Sigma x : A\ \gamma . B\ \gamma\ x \in \mathsf{Type}}$$

$$\frac{\gamma : \Gamma \vdash a : A\ \gamma \qquad \gamma : \Gamma \vdash b : B\ \gamma\ a}{\gamma : \Gamma \vdash (a, b) : \Sigma x : A\ \gamma . B\ \gamma\ x}$$

$$\frac{\gamma : \Gamma \vdash p : \Sigma x : A\ \gamma . B\ \gamma\ x}{\gamma : \Gamma \vdash \pi_1\ p : A\ \gamma} \qquad \frac{\gamma : \Gamma \vdash p : \Sigma x : A\ \gamma . B\ \gamma\ x}{\gamma : \Gamma \vdash \pi_2\ p : B\ \gamma\ (\pi_1\ p)}$$

There is one subtle challenge that arises when considering the semantic interpretation of Σ-types we have so far ignored: The Coherence Problem. Let us look above at the rules for Σ-types. First, in the formation rule, we require $B\ \gamma\ x$ to be a type family scoped by the context $\gamma : \Gamma, x : A\ \gamma$. Then, in the introduction rule, we wish to type the second argument of the pair as $\gamma : \Gamma \vdash b : B\ \gamma\ a$. Syntactically, we know that normalization will ensure that $B\ \gamma\ a$ is the same type as $(B\ \gamma\ x)[a]$ (the type $B\ \gamma\ x$ where a is substituted in for x). That being said, semantically there is no guarantee that the categorical interpretations of $B\ \gamma\ a$ will be strictly equal to the interpretation of $B\ \gamma\ x$ pulled back along the base change that substitutes in a (although they will certainly be isomorphic). This creates a problem, as in the last paragraph I claimed that the introduction rule is sound, which requires the interpretation of (a, b) to be a section of the interpretation of $\gamma : \Gamma \vdash \Sigma x : A\ \gamma . B\ \gamma\ x \in \mathsf{Type}$, and that only is the case when the interpretations of the two aforementioned types are identical. While I will not cover the various workarounds in this section, thankfully there are a number of results that solve this issue. Hoffman demonstrates how to solve this for modeling dependent type theory in locally Cartesian closed categories [39], and while more challenging for homotopical models of type theory, Lumsdaine and Warren have also developed a general solution [49] inspired by that used in Voevodsky's simplicial model of homotopy type theory [44].

Topoi

We have arrived at our destination: topoi! Let's start by going over the last missing piece of the puzzle that makes up the definition of a topos.

Definition 2.2.15. Given a category $\mathcal{C}$, a *subobject classifier* is a monomorphism $\mathsf{true} \in \mathcal{C}(1, \Omega)$. Its domain is the terminal object, and we denote the morphism by true and its codomain Ω. The subobject classifier must satisfy the following: For every monomorphism $m \in \mathcal{C}(A, B)$ in $\mathcal{C}$, there exists a unique map $\chi_m \in \mathcal{C}(B, \Omega)$ such that the following square is a pullback.

$$\begin{array}{ccc} A & \longrightarrow & 1 \\ {\scriptstyle m}\downarrow & \lrcorner & \downarrow{\scriptstyle \mathsf{true}} \\ B & \xrightarrow[\chi_m]{} & \Omega \end{array}$$

We call the morphism χ_m the *characteristic map* or *classifying map* of m. In practice, when it is clear how m describes A as a subobject of B, one often denotes the characteristic map by χ_A and omits m.

The canonical example of a subobject classifier is that in the category of sets. Thankfully, it is quite straightforward. In **Set**, the subobject classifier is given by the morphism $\mathsf{true} \in \mathbf{Set}(\{*\}, \{0, 1\})$ sending the unit element to 1. As any subobject A of B in **Set** is given by a subset of B (paired with the injection $m : A \to B$ specifying how A is a subset of B), the characteristic map χ_m is the function that sends those elements in the image of m to 1, and the rest to 0. Starting with a characteristic map $\chi \in \mathbf{Set}(B, \{0, 1\})$, let us work in the other direction and build up the corresponding monomorphism. Using the standard construction of the pullback in set, the resulting subobject is defined as

$$\{(b, *) \mid b \in B \text{ and } \chi(b) = 1\}$$

with the monomorphism sending the pair $(b, *)$ to b. As uniqueness in category theory is only ever up to isomorphism, we can just as well consider defining the subobject pullback to

instead be given by

$$\{b \in B \mid \chi(b) = 1\}$$

with the subobject monomorphism being the inclusion into B. From this, it is clear to see how maps into the subobject classifier correspond to all subsets, as quite literally the hom-set $\mathbf{Set}(B, \{0, 1\})$ is the power set of B, with the pullback picking out which subset corresponds to the given element of the power set. The characteristic maps out of B fully classify all monomorphisms into B, as again in category theory everything is only defined up to isomorphism; any subobject A given by a mono m is isomorphic to (i.e. in bijection with) the set $\{b \in B \mid \exists a \in A, m(a) = b\}$ as m is an injection.

As we now know what a subobject classifier is, we finally have defined all of the terms required to return to the definition of a topos.

Definition 2.2.1. A *topos* is a Cartesian closed category that has all finite limits and contains a subobject classifier.

Given a topos is a Cartesian closed category, we already know how it supports the basics of type theory. One would additionally hope it is a locally Cartesian closed category, thus supporting Π-types, which is thankfully also the case. A simple way to confirm this is by observing the following theorem.

Theorem 2.2.16 ([50, Chapter IV.7, Theorem 1])**.** *Given a topos $\mathcal{E}$ and an object $X \in \mathcal{E}$, the slice category $\mathcal{E}/X$ is also a topos.*

As every topos is Cartesian closed, we can trivially conclude our desired property.

Corollary 2.2.17. *Every topos is locally Cartesian closed.*

Furthermore, as topoi contain all finite limits, they certainly contain a terminal object, as is required to contain Σ-types in the internal language and thus support the basics of dependent type theory.

The additional structure present in a topos beyond a locally Cartesian closed category is that is contains a subobject classifier; with this inclusion, we can now define the dependent type theory that corresponds to the internal language of a topos.

Propositions and the Subobject Classifier

We now extend dependent type theory with a new type, Prop, the universe of logical propositions. This universe of propositions is extensional in that all proofs of the same proposition are strictly equal. As a simple example, consider the case where we have two natural numbers $x, y : \mathbb{N}$ in scope. In extensional type theories, the proposition $x = y$ exists as a type and has a code in the universe of propositions. Unlike with intensional type theories, it must be the case that any two proofs $p_1, p_2 : x = y$ are always equal. This notion of proposition (loosely) coincides with the propositions found in the proof assistant Coq [2].

The definition of the subobject classifier directly provides the rules one would expect in the internal language for the universe of propositions. In particular, the type Prop corresponds to the object Ω, propositions φ in context Γ are the morphisms $\varphi \in \mathcal{E}(\Gamma, \Omega)$, and decoding a proposition φ into a dependent type $\Gamma \vdash [\varphi] \in \mathsf{Type}$ is the object in $\mathcal{E}/\Gamma$ given by the pullback of φ along the subobject classifier morphism true.

The only rules missing at this point are that providing a grammar of how to express the logical propositions syntactically, and how each syntactic connective is modeled in the topos $\mathcal{E}$; as one would hope, the subobject classifier does indeed contain simple analogues to all standard logical connectives. To express them, the easiest approach is to utilize the bijection between morphisms into the subobject classifier and monomorphisms.

The first two propositions we will consider here are rather trivial: $\top$ and $\bot$. As $\top$ should represent the total subobject and thus correspond to the identity morphism on the context, we interpret the syntactic $\top$ as its classifying map. Similarly, as $\bot$ should be the empty subobject and thus represent the inclusion of the initial object into the context, $\bot$ is interpreted as its classifying map.

Let us now consider conjunction. We hope for the following.

$$\frac{\Gamma \vdash \varphi : \mathsf{Prop} \qquad \Gamma \vdash \psi : \mathsf{Prop}}{\Gamma \vdash \varphi \wedge \psi : \mathsf{Prop}} \qquad \frac{\Gamma \vdash p_\varphi : [\varphi] \qquad \Gamma \vdash p_\psi : [\psi]}{\Gamma \vdash (p_\varphi, p_\psi) : [\varphi \wedge \psi]}$$

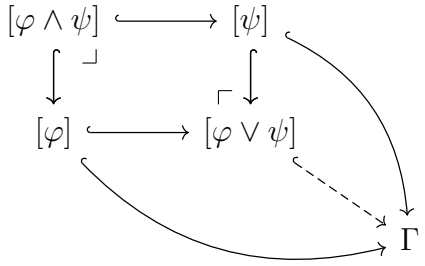
$$\frac{\Gamma \vdash p : [\varphi \wedge \psi]}{\Gamma \vdash \pi_1\ p : [\varphi]} \qquad \frac{\Gamma \vdash p : [\varphi \wedge \psi]}{\Gamma \vdash \pi_2\ p : [\psi]}$$

For the formation rule, consider two morphisms $\varphi, \psi \in \mathcal{E}(\Gamma, \Omega)$. Using the square brackets as notation to denote the subobject pullback, we can construct the following pullback.

$$\begin{array}{ccc} [\varphi \wedge \psi] & \hookrightarrow & [\psi] \\ \downarrow & \lrcorner & \downarrow \\ [\varphi] & \hookrightarrow & \Gamma \end{array}$$

As pullbacks preserve monomorphisms, all of the arrows in the diagrams are monos, and thus we can define the classifying map of the morphism from pullback into Γ as the definition of conjunction $\varphi \wedge \psi \in \mathcal{E}(\Gamma, \Omega)$, with the pullback itself being the interpretation $[\varphi \wedge \psi]$ of the conjunction proposition. The expected logical rules corresponding to conjunction follow from how it is defined as a pullback.

The logical connection of disjunction follows a similar pattern; for φ and ψ given as above, we construct $\varphi \vee \psi$ as the pushout below.



The universal property of the pushout gives us the mono including $[\varphi \vee \psi]$ into Γ induced by the inclusions of $[\varphi]$ and $[\psi]$.

$$\frac{\Gamma \vdash \varphi : \mathsf{Prop} \qquad \Gamma \vdash \psi : \mathsf{Prop}}{\Gamma \vdash \varphi \vee \psi : \mathsf{Prop}}$$

$$\frac{\Gamma \vdash p : [\varphi]}{\Gamma \vdash \iota_1\ p : [\varphi \vee \psi]} \qquad \frac{\Gamma \vdash p : [\psi]}{\Gamma \vdash \iota_2\ p : [\varphi \vee \psi]}$$

$$\frac{\Gamma, [\varphi] \vdash a_\varphi : A \qquad \Gamma, [\psi] \vdash a_\psi : A \qquad \Gamma, [\varphi \wedge \psi] \vdash a_\varphi = a_\psi}{\Gamma, [\varphi \vee \psi] \vdash (a_\varphi, a_\psi) : A}$$

When constructing a map out of a disjunction piecewise, the maps out of each disjunct must be definitionally equal on their intersection (i.e. their conjunction) to ensure the maps are coherent and thus can be combined.

Lastly, we can use the subobject classifier to model strict, extensional/judgmental equality between two terms of a type. Given a type A in context Γ, and two terms a_0 and a_1 of type A, we can define the monomorphism classified by the proposition $a_0 =_A a_1$ as the equalizer shown below.

$$[a_0 =_A a_1] \hookrightarrow \Gamma \underset{a_1}{\overset{a_0}{\rightrightarrows}} A$$

As with our other connectives, the standard rules from type theory for extensional equality with equality reflection follow from the categorical definition, depicted below.

$$\frac{\Gamma \vdash A \in \mathsf{Type} \qquad \Gamma \vdash a_0 : A \qquad \Gamma \vdash a_1 : A}{\Gamma \vdash a_0 =_A a_1 : \mathsf{Prop}} \qquad \frac{\Gamma \vdash a : A}{\Gamma \vdash \mathsf{refl}\ a : [a =_A a]}$$

$$\frac{\Gamma \vdash a_0 : A \qquad \Gamma \vdash a_1 : A \qquad \Gamma \vdash p : [a_0 =_A a_1]}{a_0 = a_1}$$

$$\frac{\Gamma \vdash a_0 : A \qquad \Gamma \vdash a_1 : A \qquad \Gamma \vdash p : [a_0 =_A a_1]}{p = \mathsf{refl}\ a_0}$$

Do note that the conclusions of the latter two rules are judgemental equalities.

Why Topoi?

While certainly it is great to see that dependent type theory can be interpreted into topoi, what motivates the clear fascination with them among type theorists? Why not only consider locally Cartesian closed categories, given they are sufficient for intensional type theory and many type theories of interest are indeed intensional? A simple answer to the latter question could be that the locally Cartesian categories one would consider happen to be topoi regardless, but thankfully I have a much more satisfying response that answers both questions: The internal language of a topos is sufficiently expressive to not only contain the dependent type theory one wishes to define, but the logic simultaneously supports that which is required for the meta-reasoning about the type theory. In other words, one can work directly in the topos, using the standard constructions from category theory to directly prove theorems about the properties of the type theory. Using this approach, one can immediately take advantage of what already is known to be true about the basic syntactic constructions of dependent type theory "for free," as from what we have shown in this section we know them to be true in *every* topos; furthermore, working in this way guarantees the results are compatible with the categorical semantics and are conveniently agnostic to any particular logical foundation of math so long as it supports defining topoi and the specific topos being considered. Studying type theory in this way depends heavily on the presence of a subobject classifier within the type theory as we will soon see, and thus justifies the need for a topos and not just a locally Cartesian closed category.

While far from being a complete exposition on the topic, let us briefly summarize the technique we leverage in this thesis. For a more detailed discourse, consider reading the research by Orton and Pitts that inspired our approach [57,58]. The premise is that we wish to define a new dependent type theory. The idea is relatively simple: Assume/require that the type theory can be modeled by some topos. As this is the case, we know the topos has at minimum the constructions corresponding to the basics of dependent type theory. For any new primitives required by the type theory, hypothesize their existence as objects and

morphisms in the category, and hypothesize anything else needed to interpret the additional constructions desired to be part of the type theory. Having listed all of the requirements needed of the topos, one then must identify a topos (or class of topoi) that contains this additional structure and supports all of the hypotheses, ensuring that the type theory indeed has a categorical interpretation and thus is consistent with respect to the topos logic. Then, using just the internal logic given by the fundamental constructions of a topos along with the new structure hypothesized, one can generally state the properties one wishes to prove about the type theory and prove them directly using the internal logic. While properties corresponding to the existence of a certain term or type are obviously a construction within the topos at large, meta-properties are often given by strict propositions, and thus are expressed by elements of the subobject classifier.

Working directly within the internal logic of a topos has an added significant benefit. The internal logic of a topos aligns nearly precisely to the logics contained in proof assistants such as Agda, given one postulates uniqueness of identity proofs and then ignores the "fancier" features. Also, one can postulate axioms within the proof assistant corresponding to the hypotheses added. Upon postulating these hypotheses, the proof assistant then acts as a synthetic setting to study the model of the type theory with the assistant's primitive type formers corresponding to the actual structure in the topos. Not only does this incredibly shallow embedding make the possibility of including machine checked proofs in this style of research more accessible, but often having the help of a proof assistant ends up being fundamentally necessary: Many of the constructions and propositions one wishes to prove include subtle details that are quite challenging to track with the human brain, and thus 1. the feedback given by the proof assistant mid-proof informing the user of precisely what goals remain makes writing the proof much faster, and 2. the type checker confirming a proof is correct results in a significantly more trustworthy end result, as without it small errors (personally) seem unavoidable.

2.3 Cubical Type Theory

Cubical type theory [7, 10, 13, 24] provides a constructive model of homotopy type theory. The main idea is to organize the higher dimensional path structure found in homotopy type theory in such that it is always represented by hypercubes. This choice of working with cubes stems from the fact that they have a number of properties that make it much easier than simplices (i.e. triangles, the default in categorical homotopy theory) for the mathematics to be done constructively, and they behave more naturally when considering the type theory computationally as a programming language.

To summarize the theory efficiently before diving into the details, let us instantiate the framework described in Section 1.2 to cubical type theory.

Semantic Structure	extend sets to cubical sets
Structural Syntax	interval type, cofibration logic, extension and glue types
The Property	Kan composition (types families that behave like spaces)
The Universe	the universe of Kan fibrations (the universe of types behaving like spaces)
The Universal Property	constructive/computational univalence

Semantic Structure As cubical type theory is a direct extension of dependent type theory, we first expand our setting from the "standard" model of dependent type theory in the topos of sets to the topos of cubical sets (i.e. the topos of presheaves over a cube category). We go into more detail of what makes up a cube category (as there are many) in Section 2.3.1.

Structural Syntax Having placed cubical type theory inside of the topos of cubical sets, we then expose the additional semantic structure now available by adding a few new items to the syntax.

The primary addition is a new type $\mathbb{I}$, called the interval type. Corresponding to the generating object of the cube category, it allows the syntax to express n-cubes as n-ary products of the type $\mathbb{I}$; furthermore, it provides syntax to access the n-cubes inside of any

type A as functions $\mathbb{I}^n \to A$ via Yoneda. Most notably, paths in a type A can now be represented as functions $\mathbb{I} \to A$.

In conjunction to the interval type itself, we add syntax for propositions pertaining to the interval type, providing a way to discuss specific boundaries of cubes. Given we have a language for describing the shapes of boundaries, we also add *extension types*, which are types that are restricted with a specification of what its inhabitants must equal along such a boundary shape; for example, the path type $\mathsf{Path}_A\,(a_0, a_1)$ is defined as a Π-type from the interval into an extension of A that specifies that when the function is evaluated at the $\mathbb{0}$ point of the interval the term returned by the function must equal a_0, and at $\mathbb{1}$ the functions must return a_1.

As a final addition to the syntax, we add a new type former for *glue types* that in effect allow one to "glue" a path between types to an equivalence of types. These new types prove to be fundamental to the construction of the universal property defined in the last step of this framework.

The Property Given the goal for cubical type theory is to be a model of homotopy type theory, we wish to classify types that behave as topological spaces. Using the new syntax for the interval type and its boundary propositions, one can define a type predicate classifying *Kan fibrations*, which informally can be thought of as types that indeed act the same as spaces. *Kan types* satisfying this predicate are automatically equipped with a structure that allows paths to be composed, including at higher dimensions.

The Universe The next step is to define a universe of Kan fibrations $\mathsf{U}_{\mathsf{Kan}}$, which one typically does using the LOPS construction [47].

The Universal Property Lastly, using all of the pieces from the previous four steps, one can classify the paths for the universe of Kan fibrations as an internal, user written term: $\texttt{univalence} : \Pi A\ B : \mathsf{U}_{\mathsf{Kan}}.\mathsf{Equiv}\ (\mathsf{Path}_{\mathsf{U}_{\mathsf{Kan}}}\ A\ B)\ (\mathsf{Equiv}\ A\ B)$. For every pair of Kan types

A and B, the type of paths between A and B is itself equivalent to the type of equivalences between A and B. Note that the definition of equivalence is given in Definition 2.1.3. Furthermore, as all of the above work is done constructively and `univalence` is written as a term inside of the type theory, it is a program that computes the equivalence and not just a property that is justified externally by the semantics.

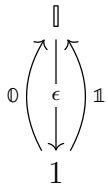
2.3.1 Cube Categories

We begin our foray into (bi)cubical type theories by first explaining what we mean by a cube category. To summarize in a sentence, a cube category is a free monoidal category (and in practice almost always a free Cartesian category) generated by an interval object $\mathbb{I}$ and a few generating morphisms. As such, they contain countably many objects $\mathbb{I}^n$ which we think of geometrically as representing the n-dimensional cube. Taking this in mind, the epimorphisms $\mathbb{I}^{n+1} \twoheadrightarrow \mathbb{I}^n$ represent the ways to surject an $n+1$-cube onto an n-cube and are called degeneracies, the monomorphisms $\mathbb{I}^n \hookrightarrow \mathbb{I}^{n+1}$ represent the injections of the n-dimensional cube into the boundaries of $n+1$-cube and are called boundary maps and the isomorphisms $\mathbb{I}^n \xrightarrow{\sim} \mathbb{I}^n$ are the symmetries of the n-cube. The variation between the different cube categories is simply the choice of which degeneracies, boundary maps, and isomorphisms are included as morphisms. Buchholtz and Morehouse present a detailed account of cube categories and their generators [16] which we use as the basis of our summary in this section.

The Cubical Interval

The one thing all cube categories have in common is that they are generated by the interval object with its endpoint morphisms. Most often we refer to the interval object as $\mathbb{I}$. Given we wish to think of this object as being an interval, we also always include two morphisms $\mathbb{0}, \mathbb{1} \in \hom(1, \mathbb{I})$—where 1 is the the nullary product (and generally the terminal object as well)—representing the inclusions of the two endpoints of the interval. We call these the *face maps*.

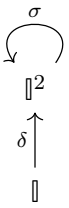
We generally also include the *degeneracy* morphism $\epsilon \in \hom(\mathbb{I}, 1)$ that is the unique map to the nullary product as a terminal object, representing the collapse of the interval to the point. The following picture summarizes this minimal choice of generators.



To ensure 1 is indeed the terminal object, we add equations saying that precomposing the degeneracy map with either endpoint inclusion is the identity on 1: $\eta \circ \mathbb{0} = \eta \circ \mathbb{1} = \mathrm{id}_1$.

Diagonals and Exchange

The first choice one makes in defining a cube category is whether it is just monoidal or also Cartesian. In particular, this comes down to whether the generators include the *exchange map* $\sigma \in \hom(\mathbb{I}^2, \mathbb{I}^2)$ that swaps the two dimensions, and the *diagonal map* $\delta \in \hom(\mathbb{I}, \mathbb{I}^2)$ that geometrically including the interval into the square along the diagonal. While certainly one can include one without the other, all of the cube categories we use in this thesis are Cartesian and thus automatically must include both.

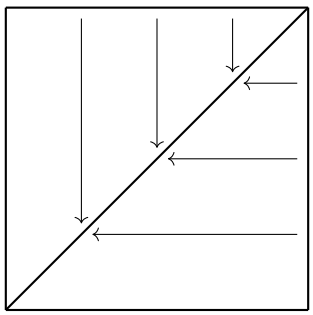


For the addition of these maps to work as intended when freely generating the category, they must be accompanied by equations dictating how they compose. In particular, we know that swapping two dimensions twice should be the identity, so $\sigma^2 = \mathrm{id}_{\mathbb{I}}$. Furthermore, to encode that σ indeed is swapping the dimensions, we define equations for how it composes with both projection maps: $\pi_1 \circ \sigma = \pi_2$ and $\pi_2 \circ \sigma = \pi_1$. We must also add equations for

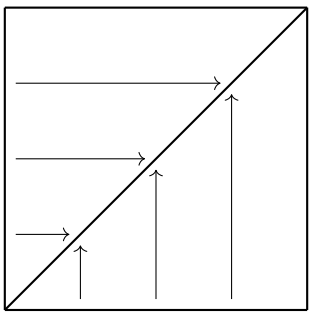
how the diagonal composes with the face maps: $\delta \circ \mathbb{0} = \mathbb{0} \times \mathbb{0}$ and $\delta \circ \mathbb{1} = \mathbb{1} \times \mathbb{1}$. Put more simply, these equations hold if and only if $\mathbb{I}^2$ is the standard binary product $\mathbb{I} \times \mathbb{I}$.

Connections

We now move onto the more interesting structural morphisms some cube categories include. The first class of morphisms are the *connections*: $\_ \wedge \_, \_ \vee \_ \in \hom(\mathbb{I}^2, \mathbb{I})$. Geometrically, the two correspond to collapsing the square onto the diagonal as shown below:



$\_ \wedge \_$



$\_ \vee \_$

These maps are accompanied by many more equations than the previous morphisms we have discussed, and are listed below. For intuition, we think of $\wedge$ as taking the minimum of the two dimensions, and $\vee$ as taking the maximum. More precisely, the connections are adding the structure of a distributive lattice bounded by $\mathbb{0}$ and $\mathbb{1}$. Taking inspiration from standard lattice structure notation, we write $x \wedge y$ for the morphism $(\_ \wedge \_) \circ (x \times y)$ where x and y are morphisms with codomain $\mathbb{I}$ (and analogously we do the same for $\_ \vee \_$).

$$
\begin{aligned}
x \wedge \mathbb{0} &= \mathbb{0} \\
x \wedge \mathbb{1} &= x \\
x \wedge x &= x \\
x \wedge y &= y \wedge x \\
x \wedge (x \vee y) &= x \\
x \wedge (y \wedge z) &= (x \wedge y) \wedge z \\
x \wedge (y \vee z) &= (x \wedge y) \vee (x \wedge z)
\end{aligned}
\qquad
\begin{aligned}
x \vee \mathbb{0} &= x \\
x \vee \mathbb{1} &= \mathbb{1} \\
x \vee x &= x \\
x \vee y &= y \vee x \\
x \vee (x \wedge y) &= x \\
x \vee (y \vee z) &= (x \vee y) \vee z \\
x \vee (y \wedge z) &= (x \vee y) \wedge (x \vee z)
\end{aligned}
$$

While most often cube categories with connections include both $\_ \wedge \_$ and $\_ \vee \_$, one can choose to have only one without the other simply by omitting the equations that involve the excluded connection.

Reversals

The last generating morphism we consider is the *reversal*. As its name suggests, it reverses the direction of the interval, swapping the two endpoints: $\rho \in \hom(\mathbb{I}, \mathbb{I})$. The first equations accompanying the reversal are straightforward: reversing twice is the same as doing nothing, so $\rho^2 = \mathrm{id}_{\mathbb{I}}$; reversing swaps the endpoints, so $\rho \circ \mathbb{0} = \mathbb{1}$ and $\rho \circ \mathbb{1} = \mathbb{0}$; reversing a degenerate interval is still degenerate, so $\eta \circ \rho = \eta$ (and thus the terminal object 1 is indeed a terminal object). Lastly, in the the presence of diagonal maps we must also add that the reversal of the diagonal is the same as the diagonal of the square in which we reverse both dimensions: $\delta \circ \rho = (\rho \times \rho) \circ \delta$.

Important Cube Categories

While there is work using more sparse collections of generators [23, 43], in this thesis every cube category considered will at minimum contain the always required required face maps, along with degeneracies, diagonals and exchange; as mentioned previously, this indicates that the cube categories are free Cartesian generated categories, and not just free monoidal. The smallest free Cartesian generated category from the interval, including just these maps, is called the *Cartesian* cube category. A model of cubical type theory defined over Cartesian cubical sets is describe in ABCFHL [7]. While our work on directed type theory is agnostic to the choice of the cube category used for the underlying homotopy structure so long as it is freely Cartesian, we implicitly work with the constraint that our work is defined using the minimal choice of the Cartesian cubes, ensuring we do not ever mistakingly use homotopical cube structure specific to the more expressive cube categories.

The second cube category we will discuss, and that appearing the most within this thesis, is the *Dedekind* cube category. In addition to the Cartesian morphisms, it also contains both connections. The primary reason we need to use this category in the setting of directed type theory is that connections allow us to define inequality: $x \leqslant y$ is the equalizer of x and $x \wedge y$ (or equivalently y and $x \vee y$). This cube category is also particularly nice to work with mathematically—particularly in the context of synthetic category theory—due to a second, strictly isomorphic definition of the category: the Dedekind cube category is the full category of **PoSet** containing all finite products (including the nullary product) of the interval poset $\mathbb{2}$. As a reminder, the interval poset is the partial order on two elements $\mathbb{0}$ and $\mathbb{1}$ where $\mathbb{0}$ is strictly less than $\mathbb{1}$. As such, we can think of the n-dimensional Dedekind cube as the product poset $\mathbb{2}^n$, and all morphisms between Dedekind cubes as the order preserving maps on the posets. Furthermore, **PoSet** itself is a full subcategory of **Cat** with its inclusion interpreting elements as object and the inequality relation as morphisms, and thus we can equivalently think of the Dedekind cubes as the full subcategory of **Cat** generated by all finite products of the interval category $\mathbb{2}$, drawn below:

$$\mathbb{0} \longrightarrow \mathbb{1}$$

As variants of the Dedekind cubes, we have the *semi-Dedekind* cube categories, also called the *semilattice* cube categories, which is the cube categories containing only one of the connections. Given there are two connections, we have both the meet-semi-Dedekind cubes (containing $\wedge$) and the join-semi-Dedekind cubes (containing $\vee$). Defining cubical type theory over these categories has been a particularly fruitful choice, as Cavallo and Sattler derived a Quillen-equivalence between the type theoretic model structure on (join-)semi-Dedekind cubical type theory and the Kan-Quillen model structure on simplicial sets (itself Quillen-equivalent to **Top**) [22].

As a final note, we will quickly define the *DeMorgan* cubical sets, which is that generated by all of the morphisms described in this section. While it has proven itself useful as the basis of the CCHM model of cubical type theory [24], we do not use it in this thesis.

Below is a table summarizing the cube categories we describe in this section.

Cube Category	**Generating Morphisms**
Cartesian	$\mathbb{0}, \mathbb{1}, \epsilon, \delta, \sigma$
semi-Dedekind	$\mathbb{0}, \mathbb{1}, \epsilon, \delta, \sigma, \vee$ (or $\wedge$)
Dedekind	$\mathbb{0}, \mathbb{1}, \epsilon, \delta, \sigma, \wedge, \vee$
DeMorgan	$\mathbb{0}, \mathbb{1}, \epsilon, \delta, \sigma, \wedge, \vee, \rho$

2.3.2 Defining Cubical Type Theory

In the internal language approach described at the end of Section 2.2.3, models of cubical type theories are described by axioms in an extensional type theory that serves as the internal language of a presheaf 1-topos. This means that we use type theoretic syntax (formalized in Agda, with postulates for function extensionality and uniqueness of identity proofs [40]) to describe constructions in cubical sets. In homotopy type theory based on Martin-Löf intensional type theory [74], all types are *fibrant*, which roughly means that all types have a transport function for paths. In the internal language approach, types instead denote "raw" cubical sets, which do not necessarily have such transport functions. Being fibrant is an internally definable structure on a type, in the style of type classes: for any type (family) A, there is another type classifying fibration structures on A.

How do we define the notion of a type being fibrant? To do so, we first need the language to discuss the boundaries of cubes. We do so by defining the *cofibrations*.

Definition 2.3.1. In cubical type theory, *cofibrations* are a special class of strict propositions that describe boundaries and subshapes of spaces.[3]

[3]While the term does originate from model category theory, and in particular the cofibrations in the type theory are the cofibrations of the model category that models the type theory, it is important to note that this *is not* the definition of a cofibration in model category theory.

In the internal language approach [57,58], an Agda type iscof identifies those propositions which are cofibrations; the axioms about cofibrations needed in [7] are shown in Figure 2.2. This hypothesizes a predicate iscof, which determines when a proposition is a cofibration, and says that cofibrations are closed under conjunction, disjunction, equality on the interval, and universal quantification over the interval. We write Ω_{cof} for the type of cofibrations, and $[\![\alpha]\!]_{\mathsf{cof}}$ for the underlying proposition of a cofibration. Here, Ω is the type of strict propositions in Type_0, i.e. types α such that $\Pi x, y : \alpha, x = y$. Conjunction $\wedge$ and universal quantification Π coincide with the standard Agda types, while we postulate a "squashed" disjunction $\vee$ where any two proofs are equal.

$$
\begin{array}{rcl}
\mathsf{iscof} & : & \Omega \to \Omega \\
\\
\Omega_{\mathrm{cof}} & : & \mathsf{Type} \\
\Omega_{\mathrm{cof}} & := & \Sigma \alpha : \Omega.\mathsf{iscof}\ \alpha \\
\\
[\![\_]\!]_{\mathsf{cof}} & : & \Omega_{\mathrm{cof}} \to \Omega \\
[\![\alpha]\!]_{\mathsf{cof}} & := & \mathsf{fst}\ \alpha \\
\\
\mathsf{iscof}\bot & : & \mathsf{iscof}\ \bot \\
\mathsf{iscof}\wedge & : & \Pi \alpha\ \beta : \Omega_{\mathrm{cof}}.\mathsf{iscof}\ [\![\alpha]\!]_{\mathsf{cof}} \wedge [\![\beta]\!]_{\mathsf{cof}} \\
\mathsf{iscof}\vee & : & \Pi \alpha\ \beta : \Omega_{\mathrm{cof}}.\mathsf{iscof}\ [\![\alpha]\!]_{\mathsf{cof}} \vee [\![\beta]\!]_{\mathsf{cof}} \\
\mathsf{iscof}\mathbb{I}\mathsf{eq} & : & \Pi i\ j : \mathbb{I}.\mathsf{iscof}(i = j) \\
\mathsf{iscof}\forall_{\mathbb{I}} & : & \Pi \alpha : \mathbb{I} \to \Omega_{\mathrm{cof}}.\mathsf{iscof}(\Pi i : \mathbb{I}.[\![\alpha\ i]\!]_{\mathsf{cof}})
\end{array}
$$

Figure 2.2: Axioms for cofibrations in cubical type theory

Having defined our cofibrations, we now define a few basic types that act as the foundation of working with this newly exposed structure.

Definition 2.3.2. Given a cofibration α, a *partial type* A is a term of the function type $[\alpha] \to \mathsf{U}$ where $[\alpha]$ is the type of witnesses to the cofibration α. We write $\alpha \vdash A$ to denote A is a partial type defined along α.

Definition 2.3.3. Given a cofibration α and a (potentially partial) type $\alpha \vdash A$, a *partial term* a is a term of the function type $[\alpha] \to A$. We write $\alpha \vdash a : A$ to denote a is a partial term of type A defined along α.

Definition 2.3.4. Given a type A, a cofibration α and a partial term $\alpha \vdash a_\alpha : A$, the *boundary type* (also called *extension type*) $A[\alpha \mapsto a_\alpha]$ is the type containing terms $a : A$ such that, when α is true, a is judgmentally equal to a_α.

For disjunctions, we use the notation $A[\alpha_1 \mapsto a_1, \alpha_2 \mapsto a_2]$ to denote the extension type $A[\alpha_1 \vee \alpha_2 \mapsto a]$ where a_x is the restriction of a to α_x. The disjunction $\alpha_1 \vee \alpha_2$ represents the pushout of the pullback of α_1 and α_2, which requires that a_1 and a_2 be strictly equal along the intersection $\alpha_1 \wedge \alpha_2$. When using the term of an extension type $x : A[\alpha \mapsto a]$, we often will omit the projection from $A[\alpha \mapsto a]$ to A and write $x : A$ as well.

As an example of how this new syntax fits together, we can use the above constructions to define the path type for cubical type theory.

$$\begin{array}{lcl} \mathsf{Path} & : & \Pi A : \mathsf{Type}.A \to A \to \mathsf{Type} \\ \mathsf{Path}\ A\ a_0\ a_1 & := & \Pi i : \mathbb{I}.A[i = \mathbb{0} \mapsto a_0, i = \mathbb{1} \mapsto a_1] \end{array}$$

In practice we generally write the type we are considering paths in as a subscript and thus we use the notation $\mathsf{Path}_A\ a_0\ a_1$ to refer to the type defined above.

At this point, we can define the properties classifying the "well-behaved" types in cubical type theory.

Definition 2.3.5. A type A has a *homogeneous composition filling structure* when it is equipped with a term witnessing the following predicate.

$$\begin{array}{lcl} \mathsf{hasHCom}\ A := & \Pi r\ r' : \mathbb{I}.\Pi \alpha : \mathsf{Cof}. \\ & \Pi t : \mathbb{I} \to [\alpha] \to A. \\ & \Pi b : A[\alpha \mapsto t\ r]. \\ & A[\alpha \mapsto t\ r', r = r' \mapsto b] \end{array}$$

This says that a type A has a homogenous composition structure when for any partial path t in A and any total point b in A at that strictly equals the path t at location r, we have a way to derive a new point in A which restricts to $t\ r'$ and b where they are defined.

In the presence of a witness $\mathsf{hcomA} : \mathsf{hcomFill}\ A$, we often write $\mathsf{hcom}_A^{z:r \to r'}\ [\alpha \mapsto t\ z]\ b$ to denote the term $\mathsf{hcomA}\ r\ r'\ \alpha\ (\lambda z.t\ z)\ b$.

Definition 2.3.6. A type family $A : \Gamma \to \mathsf{U}$ has a *coercion structure* when it is equipped with a term witnessing the predicate $\Pi p : \mathbb{I} \to \Gamma.\mathsf{hasCoe}(A \circ p)$ where hasCoe is defined below.

$$\begin{aligned}\mathsf{hasCoe}\ A := \quad & \Pi r\ r' : \mathbb{I}. \\ & \Pi b : A\ r. \\ & A\ r'[r = r' \mapsto b]\end{aligned}$$

This says that a type family $A : \Gamma \to \mathsf{Type}$ has a coercion structure when for any path p in Γ and any total point b over the path p in A at r, we can get a new point in A over p at any location r' along the path which equals b should r equal r'.

In the presence of a witness $\mathsf{coeA} : \mathsf{hcomFill}\ A$, we often write $\mathsf{coe}_A^{z:r \to r'}\ b$ to denote the term $\mathsf{coeA}\ r\ r'\ b$.

Definition 2.3.7. A type family $A : \Gamma \to \mathsf{U}$ has a *Kan composition structure* when it is equipped with a term witnessing $\Pi p : \mathbb{I} \to \Gamma.\mathsf{comFill}^{\mathbb{I}}(A \circ p)$ where $\mathsf{comFill}^{\mathbb{I}}$ is defined below.

$$\begin{aligned}\mathsf{comFill}^{\mathbb{I}}\ A := \quad & \Pi r\ r' : \mathbb{I}.\Pi \alpha : \mathsf{Cof}. \\ & \Pi t : (\Pi i : \mathbb{I}.[\alpha] \to A\ i). \\ & \Pi b : A\ r[\alpha \mapsto t\ r]. \\ & A\ r'[\alpha \mapsto t\ r', r = r' \mapsto b]\end{aligned}$$

This says that a type family $A : \Gamma \to \mathsf{Type}$ is Kan when for any path p in Γ, any partial path t over p in A and any total point b over the path p in A at r, we can get a new point in A over p at any location r' along the path, which restricts to t and b where they are defined. When a type family is equipped with a Kan composition structure, we call it a *Kan fibration* or sometimes even just a *fibration*.

In the presence of a witness $\mathsf{comA} : \mathsf{comFill}_\Gamma\ A$, we often write $\mathsf{com}_A^{z:r \to r'}\ p\ [\alpha \mapsto t\ z]\ b$ to denote $\mathsf{comA}\ p\ r\ r'\ \alpha\ (\lambda z.t\ z)\ b$.

Proposition 2.3.8 ([7, 27])**.** *A type has a* Kan composition structure *if and only if it has both a homogeneous filling structure and a coercion structure.*

Having defined the "well-behaved" types, we then group all of the Kan fibrations together into the universe of Kan fibrations, $\mathsf{U}_{\mathsf{Kan}}$, using the technique described in [47]. This universe

is closed under Π, Σ, path, identity, natural number, boolean, some higher inductive types, and the universe itself is fibrant and univalent [7, Section 3].

Lastly, we can now derive univalence for the universe $\mathsf{U}_{\mathsf{Kan}}$ as a constructive term within the type theory itself. As a reminder, univalence is a term of the following type.

$$\Pi A\ B : \mathsf{U}_{\mathsf{Kan}}.\mathsf{Equiv}\ (\mathsf{Equiv}\ A\ B)\ (\mathsf{Path}_{\mathsf{U}_{\mathsf{Kan}}}\ A\ B)$$

For a full explanation of how it is derived, one can read through [7, 24]. To summarize the main ideas, one must first understand a new type former introduced in [24] specifically for the purpose of constructing univalence: the glue type. Glue types are formed from a cofibration α, a partial type $T : \alpha \to \mathsf{Type}$, a type $B : \mathsf{Type}$, and a partial equivalence $e : \Pi p : \alpha.\mathsf{Equiv}\ (T\ p)\ B$. The type $\mathsf{Glue}[\alpha \mapsto (T, e)]B$ is isomorphic to $\Sigma t : (\llbracket \alpha \rrbracket_{\mathsf{cof}} \to T).B[\alpha \mapsto e\ t]$ but when α is satisfied $\mathsf{Glue}[\alpha \mapsto (T, e)]B$ is strictly equal to T. Assuming the input types T and B are Kan fibrations the glue type is as well. We can thus use the glue type to define the key function that underlies the construction of univalence: $\mathsf{ua} : \Pi A\ B : \mathsf{U}_{\mathsf{Kan}}.\mathsf{Equiv}\ A\ B \to \mathsf{Path}_{\mathsf{U}_{\mathsf{Kan}}}(A, B)$. The definition of ua is actually relatively simple.

$$\mathsf{ua}\ A\ B\ e\ i := \mathsf{Glue}[i = \mathbb{0} \mapsto (A, e), i = \mathbb{1} \mapsto (B, \mathrm{id}_B)]\ B$$

As with extension types (Definition 2.3.4) we simply use commas to denote we have a disjunction of cofibrations, so in particular the glue type used in the definition of ua is defined over the cofibration $i = \mathbb{0} \vee i = \mathbb{1}$. The inverse to ua is given using coercion (which we know we can always do in $\mathsf{U}_{\mathsf{Kan}}$) and ultimately the two function can be shown to form the desired equivalence, concluding the definition of univalence.

2.3.3 Key Facts and Definitions

As a final installment in this introduction to cubical type theory, below is a glossary of additional definitions and theorems that are particularly foundational and/or widely used in this thesis.

Definition 2.3.9. Given types A and B we have an *isomorphism* between A and B when we can define functions $f : A \to B$ and $g : B \to A$ such that the composition $g \circ f$ strictly equals the identity function on A and $f \circ g$ strictly equals the identity function on B. Note that this notion coincides with that of isomorphic objects in the presheaf category. We write $A \simeq B$ to denote the types are *isomorphic*

Lemma 2.3.10. *Given types A, B and C such that we have an equivalence between A and B and an isomorphism between B and C, we can construct an equivalence between A and C.*

Definition 2.3.11. A *Riehl-Shulman extension type*, often abbreviated to *RS extension type*, is a type of the form $\Pi\vec{i} : \mathbb{I}^n.A[\alpha \mapsto a]$, where the cofibration α used in the extension type is entirely scoped by the interval variables $\vec{i}$ abstracted by the Π-type. Do note that in settings with more than one interval type the codomain of the Π-type may be a mixed product of the various interval types.

Definition 2.3.12. A type A has a *contractible filling structure* when it is equipped with a term witnessing the following predicate.

$$\begin{aligned} \mathsf{cfill}\ A := \quad & \Pi\alpha : \mathsf{Cof}. \\ & \Pi a_\alpha : [\alpha] \to A. \\ & A[\alpha \mapsto a_\alpha] \end{aligned}$$

In the presence of a witness $\mathsf{cfillA} : \mathsf{cfill}\ A$, we often write $\mathsf{cfill}_A\ [\alpha \mapsto t]$ to denote $\mathsf{cfillA}\ \alpha\ t$.

The following definitions pertain to categorical models of homotopy type theory, and thus are statements based in the internal language of the topos. For more background on categorical logic and internal languages in general, see Section 2.2.

Definition 2.3.13. A type family $A : \Gamma \to \mathsf{U}$ has a *transport structure* when it is equipped with a term witnessing the following predicate.

$$\begin{aligned} \mathsf{hasTransp}\ A := \quad & \Pi p : \mathsf{Path}_\Gamma\,(\gamma_0,\ \gamma_1). \\ & A\ \gamma_0 \to A\ \gamma_1 \end{aligned}$$

Lemma 2.3.14 ([24])**.** *Assume we have a type A equipped with a homogenous composition structure. The type A is contractible if and only if it can be equipped with a contractible filling structure; furthermore, the types* $\mathsf{iscontr}\ A$ *and* $\mathsf{cfill}\ A$ *are equivalent (as they are interderivable and both are homotopy propositions).*

2.4 Directed Type Theory

Directed type theory extends homotopy type theory with a second directed notion of paths, providing a synthetic logic for studying ∞-categories as opposed to ∞-groupoids. In this thesis, we build most directly off of Riehl and Shulman's bisimplicial model of directed type theory [62]. In their work, Riehl and Shulman extend the simplicial model of homotopy type theory by adding a second copy of the simplex category to the presheaves which is then used to encode the new directed structure. From this, they then use the techniques originally developed for homotopy type theory to build up the structures and definitions required to synthetically study ∞-category theory within the type theory.

Let's first summarize the structure of bisimplicial directed type theory through the lens of the framework described in Section 1.2.

Semantic Structure	extend simplicial sets to bisimplicial sets
Structural Syntax	directed interval type, cofibration logic for new directed structure
The Properties	1. the Segal condition (types that behave like ∞-categories) 2. covariant composition (types families behaving like covariant space-indexed presheaves)
The Universe	the universe of Covariant fibrations (the universe of type families behaving like covariant space-indexed presheaves)
The Universal Property	directed univalence

Semantic Structure As an extension of homotopy type theory, bisimplicial directed type theory begins with the "standard" model of homotopy type theory in simplicial sets, and extends the semantic structure to bisimplicial sets, i.e. presheaves over the product of the simplex category with itself. The new second copy of the simplex category is ultimately used to contain the additional directed structure. For a brief introduction to the simplex category, see Section 2.4.1.

Structural Syntax The syntax is equipped with a new interval type corresponding to the directed interval, and the syntax for the cofibration logic is extended to account for propositions describing the boundaries of shapes described using the new interval type.

The Properties Unlike with the type theories described up to this point, directed type theory contains multiple properties classifying classes of types with varying structures. First is the *Segal condition*, which is a predicate on types identifying those that semantically correspond to ∞-categories. A refinement of the Segal condition is often used called the *Rezk condition*, which classifies univalent ∞-categories.

The final proposition we consider here is the *covariant condition*, which picks out the type families/fibrations whose fibers are ∞-groupoids (i.e. spaces, having trivial directed structure) and depend on the directed structure in the context covariantly. These type families are also called *left fibrations*. It is (a version of) this property that gives us the transport operation used in the motivating example in Section 1.1.

The Universe The covariant universe $\mathsf{U}_{\mathsf{cov}}$ classifies the covariant/left fibrations.

The Universal Property Cavallo, Riehl and Sattler prove that the directed path structure for the universe $\mathsf{U}_{\mathsf{cov}}$ satisfies directed univalence [21, 68]: For every pair of types A and B in $\mathsf{U}_{\mathsf{cov}}$, the type of directed paths $\mathsf{Hom}_{\mathsf{U}_{\mathsf{cov}}}\ A\ B$ is equivalent to the type of functions $A \to B$.

2.4.1 The Simplex Category

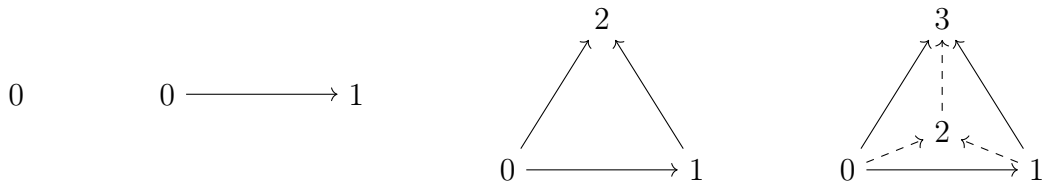
As a quick primer for those readers unfamiliar, let us define the simplex category Δ. First, from a conceptual level, the idea is that it has countably many objects, the n-th of which is called the n-simplex and corresponds to the n-dimensional triangle (or pyramid). Geometrically, one can think of the n-simplex as the structure built out of $n+1$ points, each connected to the other n points with the n-dimensional volume between all of them filled in. Specifically, the 0-simplex is the point, the 1-simplex is two points connected by a line (given by adding a new point outside of the 0-simplex and connecting it to the first point with a line), the 2-simplex is three points connected into a triangle that is then filled in (given by adding a new point outside of the 1-simplex, connecting it to the other points and filling in the area in the middle), the 3-simplex is four points connected into a pyramid with four triangular faces with its volume filled in (given by adding a new point outside of the 2-simplex, connecting it to the other three points and filling in the volume in between them all), and so on. Its utility stems from the fact that, topologically, all spaces are equivalent to one that is triangulated (i.e. broken down into a bunch of n-dimensional triangular pieces glued together along their boundaries); a triangulated spaces can be thought of as a presheaf over the simplex category, with each object being sent to the set containing all components of the space of the shape the object represents.

There are a few equivalent ways to formally define the simplex category. The first two are easy to write down, but are not particularly insightful conceptually: First, the simplex

category is that with the natural numbers as objects, and the morphisms from m to n are given by the order preserving maps from (the canonical) finite set with $m+1$ elements to that with $n+1$ elements. The second common formation of this definition is that the simplex category is the full subcategory of **PoSet** (itself a full subcategory of **Cat**) consisting of the (nonempty) finite total orders.

Having gone over the concise yet unintuitive definitions, let us now define the simplex category in a more concrete fashion. First, we denote the objects as $[n]$ for every natural number n. We still consider morphisms $\Delta([m],[n])$ as the order preserving maps from the finite set $\{0,1,\ldots,m\}$ to $\{0,1,\ldots,n\}$; that being said, we provide simple generators from which all such maps can be built from via composition. The first class of generators are the *face maps*, denoted by $\delta_i^n : [n-1] \hookrightarrow [n]$ for $0 \leq i \leq n$ (and $n \geq 1$). The map δ_i^n is the (order preserving) injection that for $j < i$ is defined as $\delta_i^n(j) = j$, and for $j \geq i$ is defined as $\delta_i^n(j) = j+1$. In other words, δ_i^n is the unique order preserving injection from $[n-1]$ to $[n]$ that skips the i-th element of $[n]$. The other class of generators needed are the *degeneracy maps*, written as $\sigma_i^n : [n+1] \twoheadrightarrow [n]$ for $0 \leq i \leq n$. The map σ_i^n is the (order preserving) surjection that for $j \leq i$ is defined as $\sigma_i^n(j) = j$, and for $j > i$ is defined as $\delta_i^n(j) = j-1$. Equivalently, σ_i^n is the sole order preserving surjection from $[n+1]$ to $[n]$ that sends two elements to i-th element of $[n]$. Two morphisms given by compositions of face maps and degeneracies are equal if the two resulting functions on finite sets are (extensionally) equal.

The benefit of this generative classification of the morphisms of the simplex category is that it provides a simple basis by which to understand the geometric interpretation of the category. To begin with, we think of the object $[n]$, the n-simplex, geometrically as the n-dimensional object built out of $n+1$ points, with each point labeled/ordered with the natural numbers 0 through $n+1$. The first four simplices are depicted below.



0-simplex 1-simplex 2-simplex 3-simplex

A morphism $f \in \Delta([m],[n])$ geometrically represents the continuous map from the m-simplex to the n-simplex induced by mapping the vertices according to the function f. For example, the face map $\delta^3_1 \in \Delta([2],[3])$ is the map including the triangle to that on the back left side of the pyramid as depicted above (and thus the triangular face of the pyramid that does not include vertex 1; the inclusion is oriented such that vertex 0 of the triangle is mapped onto vertex 0 of the pyramid, 1 goes to 2, and 2 to 3. In general, the face maps pick out one of the $(n-1)$-dimensional faces of the n-simplex by mapping the $(n-1)$-simplex onto that face in the unique orientation that preserves the total order on vertices. As a second example, the degeneracy $\sigma^1_0 \in \Delta([2],[1])$ is the map from the triangle to the line segment that sends both the vertices 0 and 1 of the triangle to the 0 end of the line segment, and sends 2 to the other side of the line segment, effectively collapsing the triangle into a line segment by squishing vertices 0 and 1 of the triangle together. This pattern generalizes such that degeneracy maps are always given by collapsing the $n+1$-simplex into the n-simplex by pinching a pair of vertices that are adjacent with respect to the order together, surjecting it onto the n-simplex in the unique way that preserves the total order on vertices. Given all maps are simply compositions of face maps and degeneracy maps, building up the geometric interpretation of an arbitrary simplicial morphism breaks down into stringing together a sequence of face inclusions and vertex collapses.

While the simplex category is obviously important to understand for the (bi)simplicial models of homotopy type theory and directed type theory, it also plays an important role in Section 4.2.2 when we define the cobar operator.

2.4.2 Defining Bisimplicial Directed Type Theory

Bisimplicial directed type theory extends homotopy type theory resulting in a logic for synthetic ∞-categories modeled in bisimplicial sets defined initially in [62]. Unpacking this one sentence summary, this means that types are represented by presheaves over $\Delta \times \Delta$ such that all types are fibrant (i.e. satisfy the Kan condition with respect to the first copy of Δ), and some presheaves/types (that in particular satisfy a certain additional property pertaining to the second copy of Δ, the Segal condition) are themselves ∞-categories, with all constructions in the logic automatically respecting and preserving their topological and categorical structure. The first copy of the simplex category is used by the presheaves to encode the topological structure of types identically to homotopy type theory, while the second new copy of the simplex category captures the directed structure of the types. In higher category theory, Segal bisimplicial sets are often used as one of the primary models of ∞-categories and their definition in bisimplicial directed type theory is precisely the same, connecting the type theory to its semantics and the surrounding body of work in higher category theory automatically; this also makes selecting bisimplicial sets as a setting for directed type theory an obvious and ideal choice from the perspective of mathematics and higher category theory. Let's now work through the basics of the theory as described in [62].

To define a type theory that interprets into bisimplicial sets, one must add a small amount of syntax to that of homotopy type theory. The primary addition is the directed interval type: $\mathbb{2}$, along with two terms $\mathbb{0}_{\mathbb{2}}$ and $\mathbb{1}_{\mathbb{2}}$ representing the two endpoints of the interval. Semantically, this type corresponds to the yoneda embedding of the 1-simplex from the second copy of the simplex category (i.e. the object $([0], [1])$).

As is always the case when adding syntax for a representable to a type theory based in homotopy theory, we also add new syntax to allow for cofibrations describing boundaries of shapes built out of the directed interval. In particular, we add a new atomic proposition for interval inequality: Given two terms x and y in $\mathbb{2}$, there is a cofibration $x \leq y$. Along with

the proposition existing itself, we know quite a few facts to hold about it.

$$\begin{aligned} x : \mathbb{2} \vdash &\quad \mathbb{0}_{\mathbb{2}} \leq x \\ x : \mathbb{2} \vdash &\quad x \leq \mathbb{1}_{\mathbb{2}} \\ x, y : \mathbb{2} \vdash &\quad x \leq y \vee y \leq x \\ \cdot \vdash &\quad \mathbb{1}_{\mathbb{2}} \leq \mathbb{0}_{\mathbb{2}} \equiv \bot \end{aligned}$$

To summarize these rules, terms of the directed interval form a nontrivial total order bounded below by $\mathbb{0}_{\mathbb{2}}$ and above by $\mathbb{1}_{\mathbb{2}}$. Semantically, this corresponds to the total order on vertices within a simplex. Having added this cofibration proposition, one can now express any representable of the new copy of the simplex category within the syntax, as the n-simplex is simply $\Sigma(x_1, \ldots, x_n) : \mathbb{2}^n . x_1 \leq \ldots \leq x_n$.

As mentioned, directed type theory is unique in classifying a few important groups of types using multiple properties. The first class is that of Segal types, consisting of those types that semantically are interpreted as ∞-categories. In order to define the Segal condition, we first must define a few special extension types. The first is the morphism type. Analogous to how path types are defined in cubical type theory, given a type A and two terms $a_0, a_1 : A$, we define the morphism type as the extension type

$$\mathsf{Hom}_A\,(a_0,\ a_1) := \Pi i : \mathbb{2}. A[i = \mathbb{0}_{\mathbb{2}} \mapsto a_0, i = \mathbb{1}_{\mathbb{2}} \mapsto a_1].$$

The second type we need is the type of directed triangles. Given a type A, three terms $a_0, a_1, a_2 : A$ and three morphisms $f : \mathsf{Hom}_A\,(a_0,\ a_1)$, $g : \mathsf{Hom}_A\,(a_1,\ a_2)$ and $h : \mathsf{Hom}_A\,(a_0,\ a_2)$, the type of triangles is given by

$$\mathsf{Hom}^2_A(f,\ g,\ h) := \Pi(i, j) : \mathbb{2}^2. i \leq j \to A[i = \mathbb{0}_{\mathbb{2}} \mapsto f, j = \mathbb{1}_{\mathbb{2}} \mapsto g, i = j \mapsto h].$$

Using these, we can define the Segal condition. In plain English, a type A is Segal if for every pair of compatible morphisms in A there is a unique (up to homotopy) triangle corresponding to their composition. Formally, we represent this predicate as follows.

$$\begin{aligned}\mathsf{isSegal}\ A := \quad & \Pi a_0, a_1, a_2 : A. \\ & \Pi f : \mathsf{Hom}_A\,(a_0,\ a_1). \\ & \Pi g : \mathsf{Hom}_A\,(a_1,\ a_2). \\ & \mathsf{iscontr}\ \Sigma h : \mathsf{Hom}_A\,(a_0,\ a_2).\mathsf{Hom}^2_A(f,\ g,\ h)\end{aligned}$$

The other class of types we will discuss are the covariant fibrations. Semantically, they correspond to left fibrations and represent space indexed covariant presheaves (and thus in particular have categorically trivial fibers). Their definition is quite straightforward: A type family $A : \Gamma \to \mathsf{U}$ is covariant if for every morphism $f : \mathsf{Hom}_\Gamma\,(\gamma_0,\ \gamma_1)$ and every term $a_0 \in A\ \gamma_0$, there is a unique morphism in A over f starting at a_0. To make this formal, we must first define the dependent version of the morphism type. For $A : 2 \to \mathsf{U}$, $a_0 : A\ \mathbb{0}_2$ and $a_1 : A\ \mathbb{1}_2$, we define the hom-over type as follows.

$$\mathsf{HomO}_A\,(a_0,\ a_1) := \Pi i : 2.A\ i[i = \mathbb{0}_2 \mapsto a_0, i = \mathbb{1}_2 \mapsto a_1].$$

Using this, we define the covariant predicate for any type family $A : \Gamma \to \mathsf{U}$.

$$\begin{aligned}\mathsf{isCov}\ A := \quad & \Pi \gamma_0, \gamma_1 : \Gamma. \\ & \Pi f : \mathsf{Hom}_\Gamma\,(\gamma_0,\ \gamma_1). \\ & \Pi a_0 : A\ \gamma_0. \\ & \mathsf{iscontr}\ \Sigma a_1 : A\ \gamma_1.\mathsf{HomO}_{A \circ f}\,(a_0,\ a_1)\end{aligned}$$

Given the fact we have multiple properties of interest, multiple universes classifying each property are also possible; that being said, the one of most interest here is the universe of covariant fibrations, $\mathsf{U}_{\mathsf{cov}}$, defined in [21, 68]. As it classifies the covariant fibrations, the types inhabiting $\mathsf{U}_{\mathsf{cov}}$ themselves have trivial categorical structure, but depend and preserve the categorical structure of their contexts covariantly.

A very exciting result about bisimplicial directed type theory is that the universe of covariant fibrations satisfies directed univalence. Directed univalence classifies the directed paths in the universe of covariant fibrations by demonstrating their equivalence to function types: For any two types A and B in $\mathsf{U}_{\mathsf{cov}}$, the type of morphisms $\mathsf{Hom}_{\mathsf{U}_{\mathsf{cov}}}\ (A,\ B)$ is equivalent to the function space $A \to B$. More formally, Cavallo, Riehl and Sattler prove that, in the model of bisimplicial sets, there is a morphism corresponding to the following term in the logic [21, 68].

$$\mathsf{directedUnivalence} : \Pi A, B : \mathsf{U}_{\mathsf{cov}}.\ \mathsf{Equiv}\ (\mathsf{Hom}_{\mathsf{U}_{\mathsf{cov}}}\ (A,\ B))\ (A \to B)$$

Unfortunately, the proof of directed univalence from Cavallo, Riehl and Sattler is not constructive; if you are interested in learning how to define directed univalence in such a way that it is constructive, keep reading! Chapter 3 defines a constructive model of directed type theory in bicubical sets, and Chapter 4 constructively defines directed univalence in the bicubical model.

Chapter 3

Bicubical Directed Type Theory

This chapter is joint work with my advisor Daniel Licata and is based primarily on of work presented at the 35th Annual ACM/IEEE Symposium on Logic in Computer Science and is published in the corresponding proceedings [83]. Versions of this work have also been presented numerous times both before and after publication including those listed here [79–81].

Bicubical directed type theory is a direct extension of cubical type theory. As such, we choose to begin by building off of Cartesian cubical type theory [7]. For our approach, we define our type theory by adding axioms to the internal language of a topos as done in [7, 57, 58]. This technique is briefly explained at the end of Section 2.2.3. The resulting model can then be given semantics by interpreting into any topos satisfying the axioms. In our case, we choose to interpret this theory into (Cartesian $\times$ Dedekind)-bicubical sets.

One key fact to realize from this approach is that the resulting model can be interpreted into any topos that satisfies the axioms; by starting with Cartesian cubical type theory, we insure that no axioms specific to more richly structured cube categories are required in our work, and thus bicubical directed type theory remains agnostic to the choice of underlying cubical type theory so long as it satisfies the minimal Cartesian cube axioms (described in Figure 3.1). To emphasize this fact, we write $\square_{\text{path}}$ to represent some arbitrary Cartesian cube category with an interval.

3.1 Axioms for Bicubical Type Theory

We begin by working inside of a topos, with the intention of it being a presheaf topos over a product of two cube categories. The first cube category will be used to contain the undirected homotopical structure, and the second will contain the directed categorical structure. To encode the homotopical cube structure, we add the axioms listed in Figure 3.1. We also add axioms describing a subobject of the subobject classifier which will represent the cofibrations in our type theory. These cofibration axioms are shown in Figure 3.2. For the resulting model to be constructive, we restrict the class of cofibration propositions that can be checked algorithmically. In order to construct univalence and directed univalence we require that our class of cofibrations contains those given by equality of interval variables, is closed under universal quantification by the intervals $\mathbb{I}$ and $\mathbb{2}$, contains the empty proposition $\bot$, and that it is closed under pullbacks (i.e. conjunction: $\wedge$) and pushouts of pullbacks (i.e. disjunction: $\vee$). As all of this is the same as what is done for cubical type theory, a more detailed explanation of the cofibration axioms is available in Section 2.3.

$$
\begin{array}{rlrl}
\mathbb{I} : & \mathsf{Type} & \mathsf{iscof}\mathbb{I}\mathsf{eq} : & \Pi i\ j : \mathbb{I}.\mathsf{iscof}\ (i = j) \\
\mathbb{0}_{\mathbb{I}} : & \mathbb{I} & \mathsf{iscof}\forall_{\mathbb{I}} : & \Pi\alpha : \mathbb{I} \to \Omega_{\mathrm{cof}}. \\
\mathbb{1}_{\mathbb{I}} : & \mathbb{I} & & \mathsf{iscof}(\ \Pi i : \mathbb{I}.[\![\alpha\ i]\!]_{\mathsf{cof}}) \\
\mathbb{I}\mathsf{neq} : & \mathbb{0}_{\mathbb{I}} = \mathbb{1}_{\mathbb{I}} \to \bot & \mathbb{I} & \text{ is tiny}
\end{array}
$$

Figure 3.1: Axioms for the undirected interval

In Figure 3.3, we describe the additional axioms needed to describe a directed type theory based on bicubical sets. Riehl and Shulman [62] extend homotopy type theory with a directed interval $\mathbb{2}$, which is used to describe simplices (point, line, triangle, tetrahedron, etc.) by inequality formulas—e.g. the triangle is the top half of the square $x : \mathbb{2}, y : \mathbb{2} \mid y \leq x$. For more background on their approach to directed type theory, see Section 2.4. Here, we extend cubical type theory, which already has an interval for paths $\mathbb{I}$, with a second *directed interval* $\mathbb{2}$. On the one hand, we would like the directed interval to support a constructive definition of a universe of covariant fibrations using the LOPS construction [47], which relies on the

$$
\begin{array}{rcl}
\mathsf{iscof} & : & \Omega \to \Omega \\
\Omega_{\mathrm{cof}} & : & \mathsf{Type} \\
\Omega_{\mathrm{cof}} & := & \Sigma \alpha : \Omega.\mathsf{iscof}\ \alpha \\
[\![\_]\!]_{\mathsf{cof}} & : & \Omega_{\mathrm{cof}} \to \Omega \\
[\![\alpha]\!]_{\mathsf{cof}} & := & \mathsf{fst}\ \alpha \\
\mathsf{iscof}\bot & : & \mathsf{iscof}\ \bot \\
\mathsf{iscof}\wedge & : & \Pi \alpha\ \beta : \Omega_{\mathrm{cof}}.\mathsf{iscof}\ [\![\alpha]\!]_{\mathsf{cof}} \wedge [\![\beta]\!]_{\mathsf{cof}} \\
\mathsf{iscof}\vee & : & \Pi \alpha\ \beta : \Omega_{\mathrm{cof}}.\mathsf{iscof}\ [\![\alpha]\!]_{\mathsf{cof}} \vee [\![\beta]\!]_{\mathsf{cof}} \\
\mathsf{strictification} & : & \Pi \alpha : \Omega_{\mathrm{cof}}. \\
 & & \Pi A : [\![\alpha]\!]_{\mathsf{cof}} \to \mathsf{Type}. \\
 & & \Pi B : \mathsf{Type}. \\
 & & \Pi s : \Pi x : [\![\alpha]\!]_{\mathsf{cof}}.\mathsf{Iso}\ (A\ x)\ B. \\
 & & \Sigma B' : \mathsf{Type}[\alpha \mapsto A].\mathsf{Iso}\ B'\ B[\alpha \mapsto s]
\end{array}
$$

Figure 3.2: Cofibration axioms

interval being tiny (exponentiation by the interval has a right adjoint), which is not true for the interval in simplical sets. On the other hand, we would like this interval to support the body of definitions that are made in [62], where triangular shapes are sometimes important—e.g. a type is a category (roughly) if, given any two composable directed morphisms, there is a unique composite morphism and a triangle relating them. We can accomplish both of these using the Dedekind cube category $\square_{\mathrm{Ded}}$, in which the interval is defined to be a distributive lattice, where the lattice meet and join are referred to as *connections*. Using the lattice structure, we can define $x \leq y$ as $x = x \sqcap y$ (x is the min of x and y). To learn more about the Dedekind cubes—along with cube categories more generally—check out Section 2.3.1.

In Figure 3.3, the first group of axioms describe the Dedekind cubes: the interval has two disjoint points; there are meet and join operations satisfying the distributive lattice laws. Relative to the simplex category axioms, this omits the requirement that $\leq$ is a *total* order ($x \leq y \vee y \leq x$ for all x and y, which allows for defining a square in a type A by pasting together two triangles in A on the diagonal). This omission does not seem to be a problem: first, we have checked that most of the definitions that use totality in [62] in fact only use it via the connections; second, many internal constructions involve Segal or discrete types,

$$
\begin{aligned}
2 &: \mathsf{Type} \\
\mathbb{0}_2 &: 2 \\
\mathbb{1}_2 &: 2 \\
2\mathsf{neq} &: \mathbb{0}_2 = \mathbb{1}_2 \to \bot
\end{aligned}
$$

$$
\begin{aligned}
\_ \sqcap \_ &: 2 \to 2 \to 2 \\
\sqcap\mathbb{0}_2 &: \Pi x : 2.x \sqcap \mathbb{0}_2 = \mathbb{0}_2 \\
\sqcap\mathbb{1}_2 &: \Pi x : 2.x \sqcap \mathbb{1}_2 = x \\
\sqcap\mathsf{comm} &: \Pi x\ y : 2.x \sqcap y = y \sqcap x \\
\sqcap\mathsf{idem} &: \Pi x : 2.x \sqcap x = x \\
\sqcap\mathsf{assoc} &: \Pi(x, y, z : 2). \\
&\quad ((x \sqcap y) \sqcap z) = (x \sqcap (y \sqcap z))
\end{aligned}
\qquad
\begin{aligned}
\_ \sqcup \_ &: 2 \to 2 \to 2 \\
\sqcup\mathbb{0}_2 &: \Pi x : 2.x \sqcup \mathbb{0}_2 = x \\
\sqcup\mathbb{1}_2 &: \Pi x : 2.x \sqcup \mathbb{1}_2 = \mathbb{1}_2 \\
\sqcup\mathsf{comm} &: \Pi x\ y : 2.x \sqcup y = y \sqcup x \\
\sqcup\mathsf{idem} &: \Pi x : 2.x \sqcup x = x \\
\sqcup\mathsf{assoc} &: \Pi(x, y, z : 2). \\
&\quad ((x \sqcup y) \sqcup z) = (x \sqcup (y \sqcup z))
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{abs}_0 &: \Pi x\ y : 2.x \sqcap (x \sqcup y) = x \\
\mathsf{abs}_1 &: \Pi x\ y : 2.x \sqcup (x \sqcap y) = x
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{iscof}2\mathsf{eq} &: \Pi i\ j : 2.\mathsf{iscof}\ (i = j) \\
\mathsf{iscof}\forall_2 &: \Pi\alpha : 2 \to \Omega_{\mathrm{cof}}. \\
&\quad \mathsf{iscof}(\ \Pi i : 2.[\![\alpha\ i]\!]_{\mathsf{cof}}) \\
2 &\ \text{is tiny} \\
\Delta^2 &\ \text{is tiny}
\end{aligned}
\qquad
\begin{aligned}
\mathbb{I}\mathsf{to}2\mathsf{triv} &: \Pi p : \mathbb{I} \to 2.\Pi x : \mathbb{I}. \\
&\quad p\ x = p\ \mathbb{0}_{\mathbb{I}} \\
2\mathsf{mono} &: \Pi p : 2 \to 2.\Pi x\ y : 2. \\
&\quad x = x \sqcap y \to p\ x = p(x) \sqcap p(y)
\end{aligned}
$$

Figure 3.3: Axioms for the directed interval

which satisfy totality weakly (with a uniqueness principle up to paths not strict equality); third, should one need to, one can internally carve out the simplices via a sheaf condition to state theorems about types that do satisfy totality [17, 73].

The next two axioms in Figure 3.3, $\mathsf{iscof}2\mathsf{eq}$ and $\mathsf{iscof}\forall_2$, say that equality on the directed interval, and Π over the directed interval, are cofibrations, extending the class of shapes that can be filled in Kan types. For example, $\mathsf{iscof}2\mathsf{eq}$ allows converting a path to a morphism in a Kan type: given $p : \mathbb{I} \to A$, we make $2 \to A$ by the Kan composition

$$
\begin{aligned}
\mathsf{path\text{-}to\text{-}hom}(p) := \lambda(x : 2).\mathsf{com}_A^{z:\mathbb{0}_{\mathbb{I}} \to \mathbb{1}_{\mathbb{I}}}[&x = \mathbb{0}_2 \mapsto p(\mathbb{0}_{\mathbb{I}}) \\
&, x = \mathbb{1}_2 \mapsto p(z)](p(\mathbb{0}_{\mathbb{I}}))
\end{aligned}
$$

which has endpoints $p(\mathbb{0}_{\mathbb{I}})$ at $\mathbb{0}_2$ and $p(\mathbb{1}_{\mathbb{I}})$ at $\mathbb{1}_2$. $\mathsf{iscof}\forall_2$ is used to show that glue types [24] (where the function is an equivalence) are covariant families, which is the key lemma for

showing that $\mathsf{U}_{\mathsf{cov}}$ is path-univalent. Tininess of 2 is needed for building $\mathsf{U}_{\mathsf{cov}}$, while tininess of Δ^2 (defined as $\Sigma(x, y) : 2 \times 2.y \leq x$) is needed for $\mathsf{U}_{\mathsf{inner}}$ [47].

The final two axioms, $\mathbb{I}\mathsf{to2triv}$ and $2\mathsf{mono}$, are used in Chapter 4 in the construction of directed univalence. $\mathbb{I}\mathsf{to2triv}$ says that any map from the undirected interval to the directed interval is constant. The second axiom says that any function $2 \to 2$ is monotonic—if $x \leq y$ then $p(x) \leq p(y)$—which, for example, rules out the existence of a reversal $1 - x$.

3.1.1 Soundness in Bicubical Sets

Extensional type theory with the axioms in Figure 3.3 can be interpreted in bicubical sets, the presheaf topos $\mathbf{Set}^{\square^{\mathrm{op}}_{\mathrm{path}} \times \square^{\mathrm{op}}_{\mathrm{Ded}}}$.

Types denotes presheaves in this topos, and have both a path structure, encoded in the Cartesian cubes, and a directed morphism structure, residing in the Dedekind cubes. We sometimes refer to the objects of the product cube category $\square_{\mathrm{path}} \times \square_{\mathrm{Ded}}$ by the natural numbers describing how many free interval variables occur (which is the dimension of the object)—i.e., we use the pair (m, n) to denote the bicube given by the m dimension Cartesian cube paired with the n dimension Dedekind cube. Thus, given a type/presheaf A, the set $A(0, 0)$ is its points, $A(1, 0)$ is paths between points, $A(0, 1)$ is directed morphisms between points, $A(1, 1)$ is squares with one side directed and the other not, and so on. The type $\mathbb{I}$ is the Yoneda embedding of the $(1, 0)$-bicube, and 2 is that of the $(0, 1)$-bicube. Because the Yoneda embedding preserves products, we can describe the representable functor on every bicube using products of these two objects: the internal language type $\mathbb{I}^m \times 2^n$ is interpreted as $\mathbf{y}_{\mathbb{I}^m} \times \mathbf{y}_{2^n} \cong \mathbf{y}_{(\mathbb{I}^m, 2^n)}$. By the Yoneda lemma, maps $\mathbb{I}^m \times 2^n \to A$ are thus naturally isomorphic to the set $A(m, n)$ given by evaluating A at the the (m, n)-bicube.

The axioms we add to the model are nearly identical to the generators needed to define each cube category as a free Cartesian category, as is described in [16] and summarized in Section 2.3.1. In the case of the directed interval 2, while we list the axiom corresponding to the associativity generator from the Dedekind cubes for the sake of completeness, we

never encountered a need for it in any of our proofs. We also added two axioms beyond the generators—namely $\mathbb{I}$to2triv and 2mono—as they prove to be necessary in our constructions and, as we will see shortly, they are easily derivable in the category of bicubical sets.

Theorem 3.1.1. *Let $\square_{\text{path}}$ be a cartesian category with an interval object $\mathbb{I}$ with two distinct endpoints $\mathbb{0}_{\mathbb{I}}$ and $\mathbb{1}_{\mathbb{I}}$ in $1 \to_{\square_{\text{path}}} \mathbb{I}$, such that for any object C, the hom-set $C \to_{\square_{\text{path}}} \mathbb{I}$ has decidable equality. Then the axioms for cartesian cubical type theory from [7, Section 3.1.1] and reproduced in Figures 3.1 and 3.2, and the axioms for bicubical directed type theory in Figure 3.3 are true in* $\mathbf{Set}^{\square_{\text{path}}^{\text{op}} \times \square_{\text{Ded}}^{\text{op}}}$.

We prove Theorem 3.1.1 with the lemmas presented in the remainder of Section 3.1.

Lemma 3.1.2. *The axioms from Cartesian cubical type theory [7, Section 3.1.1]—listed in Figures 2.2 and 3.1—pertaining to the interval $\mathbb{I}$ are all true in the category of bicubical sets, and the cofibrations for $\mathbb{I}$ all factor through the object of decidable sieves.*

Proof. In $\mathbf{Set}^{\square_{\text{path}}^{\text{op}} \times \square_{\text{Ded}}^{\text{op}}}$, we interpret $\mathbb{I}$ as $\mathbf{y}(\mathbb{I}^1, 2^0)$, i.e. the pair of the interval in $\square_{\text{path}}$ and the terminal object of $\square_{\text{Ded}}$. $\mathbb{0}_{\mathbb{I}} : \mathbb{I}$ and $\mathbb{1}_{\mathbb{I}} : \mathbb{I}$ represent (the Yoneda embedding of) the corresponding maps in $\square_{\text{path}}$, paired with the unique map into the terminal object. These satisfy $\mathbb{0}_{\mathbb{I}} \neq \mathbb{1}_{\mathbb{I}} : \mathbb{I}$ because the $\square_{\text{path}}$ components are distinct.

For the strictification axiom, [58, Theorem 8.4] shows that strictification holds constructively in any presheaf topos if the cofibrations are a subset of the decidable sieves. The cofibrations are interpreted as subobjects (conjunction, disjunction, universal quantification, equality) that exist in any topos, so all there is to show is that cofibrations are decidable sieves, i.e. that $\mathtt{Cof} \hookrightarrow \Omega$ factors $\mathtt{Cof} \hookrightarrow \Omega_{dec} \hookrightarrow \Omega$ through the presheaf of decidable sieves Ω_{dec} [58, Definition 8.3]. By the argument in [7, Section 3.2], one interpretation satisfying this is to take $\mathtt{Cof}$ to be Ω_{dec} itself: closure under $\Pi_{\mathbf{y}X}$ for any representable follows if the base category has finite products, which $\square_{\text{path}} \times \square_{\text{Ded}}$ does (given componentwise), and for $=_{\mathbb{I}}$ we need that maps from any object into $(\mathbb{I}^1, 2^0)$ have decidable equality, which is assumed.

For the approach we use to construct universes, [47, Remark 4.1] argue that the crisp type theory used in agda−flat can be interpreted in any presheaf topos on a base category with a terminal object, which is $(\mathbb{I}^0, 2^0)$. Tininess (right adjoint to exponentiation by $\mathbb{I}$) follows because in presheaves on a base category with finite products, all representables are tiny, and $\mathbb{I}$ is interpreted as $\mathbf{y}(\mathbb{I}^1, 2^0)$. ⊠

Lemma 3.1.3. *The axioms in Figure 3.3 defining the type $\mathbb{2}$ and its connections are sound in the interpretation into bicubical sets.*

Proof. $\mathbb{2}$ is interpreted as $\mathbf{y}(\mathbb{I}^0, 2^1)$. The connection axioms are the axiomatic presentation of the interval in $\square_{\text{Ded}}$ [16], and therefore are true for the Yoneda embedding of the directed interval. ⊠

Lemma 3.1.4. *The directed interval $\mathbb{2}$ is tiny.*

Proof. Analogously to $\mathbb{I}$, this follows because the base category we take our presheaves over has finite products and $\mathbb{2}$ is a representable presheaf. ⊠

Lemma 3.1.5. *The directed triangle Δ^2 is tiny.*

Proof. To prove Δ^2 to be tiny, we first demonstrate that the idempotent completion of $\square_{\text{Ded}}$ is closed under binary products. Thus, consider two representables in the idempotent completion of $\square_{\text{Ded}}$, A and B. For this proof, we will utilize the notion of the idempotent completion as that generated as a full subcategory of $\widehat{\square_{\text{Ded}}}$, and thus A and B are presheaves. As A and B are contained in the idempotent completion of $\square_{\text{Ded}}$, we know there exists objects $\tilde{A}$ and $\tilde{B}$ in $\square_{\text{Ded}}$ along with split epimorphisms $\alpha \in \widehat{\square_{\text{Ded}}}(\mathbf{y}\tilde{A}, A)$ and $\beta \in \widehat{\square_{\text{Ded}}}(\mathbf{y}\tilde{B}, B)$. Let $\tilde{\alpha}$ and $\tilde{\beta}$ be the splitting monomorphisms of α and β. With these components, we construct the following diagram in $\widehat{\square_{\text{Ded}}}$.

$$\begin{array}{ccc} \mathbf{y}\tilde{A} \times \mathbf{y}\tilde{B} & \longrightarrow & \mathbf{y}\tilde{A} \times \mathbf{y}\tilde{B} \\ & {\scriptstyle \alpha\times\beta} \searrow \quad \nearrow {\scriptstyle \tilde{\alpha}\times\tilde{\beta}} & \\ & A \times B & \end{array}$$

As $\square_{\text{Ded}}$ is closed under binary products, we know $\mathbf{y}\tilde{A} \times \mathbf{y}\tilde{B}$ is a representable functor. Furthermore, by the properties of the binary product we can see that the morphism $\alpha \times \beta$ is an epimorphism split by $\tilde{\alpha} \times \tilde{\beta}$, concluding their composition is an idempotent morphism in $\square_{\text{Ded}}$ (modulo Yoneda) and thus the product $A \times B$ is itself an element of the idempotent completion of $\square_{\text{Ded}}$.

From [15, Theorem 1], we know the category of presheaves over the idempotent completion of a category is equivalent to the category of presheaves over the original category. Given Δ^2 is an element of the idempotent completion of $\square_{\text{Ded}}$ (via the morphism sending (x, y) to $(x, x \wedge y)$), and the idempotent completion is closed under binary products, Δ^2 is a tiny object in the category of presheaves over the idempotent completion of $\square_{\text{Ded}}$. We can then utilize the equivalence of categories to conclude that the object Δ^2 is also tiny in the category $\widehat{\square_{\text{Ded}}}$. ⊠

Lemma 3.1.6. *The cofibrations pertaining to the directed interval $\mathbb{2}$ factor through Ω_{dec}.*

Proof. The same argument as for $=_{\mathbb{I}}$ and $\forall_{\mathbb{I}}$ applies: closure under $\forall$ holds for any representable because of finite products in $\square_{\text{path}} \times \square_{\text{Ded}}$ [7, Section 3.2], and maps into $(\mathbb{I}^0, \mathbb{2}^1)$ in $\square_{\text{path}} \times \square_{\text{Ded}}$ have decidable equality, because equality in homsets in $\square_{\text{Ded}}$ is decidable. ⊠

Lemma 3.1.7. *The axiom* $\mathbb{2}$mono *is true in the category of bicubical sets.*

Proof. The goal is an equality of morphisms in $X \to_{\square_{\text{path}} \times \square_{\text{Ded}}} (\mathbb{I}^0, \mathbb{2}^1)$, but since $\mathbb{I}^0$ is terminal, the $\square_{\text{path}}$ components are trivially equal, and it suffices to check

> For all $C \in \square_{\text{Ded}}$, $f : (C \times \mathbb{2} \to_{\square_{\text{Ded}}} \mathbb{2})$, and $x, y : (C \to_{\square_{\text{Ded}}} \mathbb{2})$, if $x =_{(C \to_{\square_{\text{Ded}}} \mathbb{2})} \langle x, y \rangle; \sqcap$ then $\langle id, x \rangle; f =_{(C \to_{\square_{\text{Ded}}} \mathbb{2})} \langle \langle id, x \rangle; f, \langle id, y \rangle; f \rangle; \sqcap$

Because $\square_{\text{Ded}}$ can be presented as the free cartesian category on some generators, we can view its morphisms as syntax in its internal simple type theory (with product types), and do induction on the syntax. Thus, consider it to be presented by a judgment $x_i : \mathbb{2} \vdash f : \mathbb{2}$, where the constructors for f are $\_ \sqcap \_, \_ \sqcup \_, \mathbb{0}_{\mathbb{2}}, \mathbb{1}_{\mathbb{2}}$ and variables from the context, with

the equational theory of a distributed lattice. We assume an arbitrary context Ψ and terms $\Psi \vdash u : \mathbb{2}$ and $\Psi \vdash v : \mathbb{2}$ and $\Psi, z : \mathbb{2} \vdash f : \mathbb{2}$, such that $u = u \sqcap v$. We need to show that $f[u/z] = f[u/z] \sqcap f[v/z]$ according to the equational theory. The proof is by induction on f.

- In the case where f is z, the result is the assumption.
- In the case where f is some other variable x or $\mathbb{0}_{\mathbb{2}}$ or $\mathbb{1}_{\mathbb{2}}$ (write this as k), $f[-] = k$, so idempotence of $\sqcap$ gives the result.
- In the case where f is $f_1 \sqcap f_2$, we get $f_1[u/z] = f_1[u/z] \sqcap f_1[v/z]$ and $f_2[u/z] = f_2[u/z] \sqcap f_2[v/z]$ and we need to show

$$\begin{aligned} &f_1[u/z] \sqcap f_2[u/z] = \\ &\quad (f_1[u/z] \sqcap f_2[u/z] \sqcap f_1[v/z] \sqcap f_2[v/z]) \end{aligned}$$

  which follows from the inductive hypotheses and associativity/commutativity.

- In the case where f is $f_1 \sqcup f_2$, we get $f_1[u/z] = f_1[u/z] \sqcap f_1[v/z]$ and $f_2[u/z] = f_2[u/z] \sqcap f_2[v/z]$ and we need to show

$$\begin{aligned} &f_1[u/z] \sqcup f_2[u/z] = \\ &\quad (f_1[u/z] \sqcup f_1[v/z]) \sqcap (f_2[u/z] \sqcup f_2[v/z]) \end{aligned}$$

  Multiplying out the right hand side gives

$$\begin{aligned} &(f_1[u/z] \sqcap f_2[u/z]) \sqcup (f_1[v/z] \sqcap f_2[u/z]) \sqcup \\ &(f_1[u/z] \sqcap f_2[v/z]) \sqcup (f_1[v/z] \sqcap f_2[v/z]) \end{aligned}$$

  By the inductive hypothesis this is

$$(f_1[u/z]) \sqcup (f_1[v/z] \sqcap f_2[u/z]) \sqcup (f_1[u/z] \sqcap f_2[v/z]) \sqcup (f_1[v/z])$$

  and then absorption ($x \sqcup (x \sqcap y) = x$) gives the result.

$\boxtimes$

Lemma 3.1.8. *The axiom* 2to𝕀triv *is true in the category of bicubical sets.*

Proof. The internal statement of the axiom externalizes to

For all $C \in \square_{\text{path}} \times \square_{\text{Ded}}$,
$f : (C \times (\mathbb{I}^1, 2^0)) \to_{\square_{\text{path}} \times \square_{\text{Ded}}} (\mathbb{I}^0, 2^1)$, and
$x : C \to_{\square_{\text{path}} \times \square_{\text{Ded}}} (\mathbb{I}^1, 2^0)$,
$\langle id, x \rangle ; f =_{(C \to_{\square_{\text{path}} \times \square_{\text{Ded}}} (\mathbb{I}^0, 2^1))} \langle id, (\mathbb{O}_{\mathbb{I}}, !) \rangle ; f.$

Since $\mathbb{I}^0$ is terminal, it suffices to check the maps in $\square_{\text{Ded}}$, but we have

$$f_{\square_{\text{Ded}}} : (C_{\square_{\text{Ded}}} \times 2^0) \to_{\square_{\text{Ded}}} 2^1$$
$$x_{\square_{\text{Ded}}} : C_{\square_{\text{Ded}}} \to_{\square_{\text{Ded}}} 2^0$$
$$\langle id, x \rangle ; f =_{(C_{\square_{\text{Ded}}} \to_{\square_{\text{Ded}}} 2^1)} \langle id, ! \rangle ; f$$

which is true because 2^0 is terminal. ⊠

3.2 The "Types" of Types

Riehl-Shulman [62] define many properties of types, such as being discrete (the directed morphism structure is trivial), being Segal/a category (directed morphisms have unique composites), and being a covariant discrete fibration (a family of discrete types with a "forward" transport along morphisms), and being a contravariant discrete fibration (a family of types with a "backward" transport along morphisms).

In this section we show that some of these key definitions and lemmas of [62] can be made in this setting, and draw some correspondences with cubical filling problems. As many of the definitions use the directed morphism type, we begin with its definition before moving into the many classes of types that make up the basis of working within directed type theory.

3.2.1 Directed Morphism Type

Given a type A and two terms a_0 and a_1, the type of directed morphisms from a_0 to a_1 is

$$\mathsf{Hom}_A\,(a_0, a_1) := \Pi i : 2.A[i = \mathbb{0}_2 \mapsto a_0, i = \mathbb{1}_2 \mapsto a_1]$$

Compare with the cubical path type $\mathsf{Path}_A\,(a_0, a_1) := \Pi i : \mathbb{I}.A[i = \mathbb{0}_\mathbb{I} \mapsto a_0, i = \mathbb{1}_\mathbb{I} \mapsto a_1]$. We can also define dependent morphism types where, instead of a fixed type, we consider a family of types indexed by the corresponding interval type: Given a type family $A : 2 \to \mathsf{Type}$ and two terms $a_0 : A\ \mathbb{0}_2$ and $a_1 : A\ \mathbb{1}_2$, the type of directed paths from a_0 to a_1 is

$$\mathsf{DHom}_A\,(a_0, a_1) := \Pi i : 2.(A\ i)[i = \mathbb{0}_2 \mapsto a_0, i = \mathbb{1}_2 \mapsto a_1]$$

In [62], morphism types are defined as extension types, which combine quantification over interval variables and extension constraints, restricted in such a way that an extension type is always fibrant. Here, we must show that the morphism type is Kan:

Theorem 3.2.1 (`universe.Hom.Hom-code-universal`)**.** *There is a code*

$$\mathsf{dhom} : \Pi(A : 2 \to \mathsf{U}_{\mathsf{Kan}}).A(\mathbb{0}_2) \to A(\mathbb{1}_2) \to \mathsf{U}_{\mathsf{Kan}}$$

such that $\mathsf{El}(\mathsf{dhom}(A, a_0, a_1)) = \mathsf{DHom}_{\mathsf{El} \circ A}\,(a_0, a_1)$

Proof. This argument is the same as the argument for Kan path types [7, 24], taking advantage of the fact that, for a directed interval variable x, $x = \mathbb{0}_2$ and $x = \mathbb{1}_2$ are cofibrations.

To make a code for the dependent hom type $\mathsf{DHom}_{\mathsf{El} \circ A}\,(a_0, a_1)$, we in particular must construct a Kan composition structure for it assuming one exists for the type family A (encoded by the fact the family A lands in $\mathsf{U}_{\mathsf{Kan}}$). Thus, given the context Γ, we have that $A : \Gamma \times 2 \to \mathsf{U}_{\mathsf{Kan}}$, and we can assume a witness $\mathsf{comA} : \mathsf{comFill}\ (\mathsf{El} \circ A)$. We now construct the function $\mathsf{comDHomA} : \mathsf{comFill}\ \mathsf{DHom}_{\mathsf{El} \circ A}\,(a_0, a_1)$ for $\mathsf{DHom}_{\mathsf{El} \circ A}\,(a_0, a_1) : \Gamma \to \mathsf{U}$. Given a path p in Γ,

any two interval variables $r,\ r' : \mathbb{I}$, a cofibration α scoped by Γ, tube $t : (x : \mathbb{I}) \to [\![\alpha]\!]_{\mathsf{cof}} \to \mathsf{DHom}_{(\mathsf{El} \circ A)\ (p\ x)}\,(a_0\ (p\ x), a_1\ (p\ x))$ and base $b : \mathsf{DHom}_{(\mathsf{El} \circ A)\ (p\ r)}\,(a_0\ (p\ r), a_1\ (p\ r))[\alpha \mapsto t\ r]$, we define the body of $\mathtt{comDHomA}\ p\ r\ r'\ \alpha\ t\ b : \mathsf{DHom}_{(\mathsf{El} \circ A)\ (p\ r')}\,(a_0\ (p\ r'), a_1\ (p\ r'))[\alpha \mapsto t\ r', r = r' \mapsto b]$ by Kan composition in A as shown below.

$$\begin{aligned}\mathtt{comDHomA}\ p\ r\ r'\ \alpha\ t\ b := \lambda x : 2.\mathsf{com}_A^{z : r \to r'}[&\alpha \mapsto t\ z\ x\\ &,\ x = \mathbb{0}_2 \mapsto a_0\ (p\ z)\\ &,\ x = \mathbb{1}_2 \mapsto a_1\ (p\ z)]\ (b\ x)\end{aligned}$$

$\boxtimes$

3.2.2 Discrete Types

The types with trivial directed morphism structure play the role of "sets" in directed type theory, and are the elements of the universe $\mathsf{U}_{\mathsf{cov}}$ of "sets" and functions that we define below. We use quotation marks as, given directed type theory is built on top of HoTT, the discrete types are actually the ∞-groupoids and thus still potentially have nontrivial homotopical structure. From this perspective, one can think of the discrete types as the embedding of HoTT inside of directed type theory.

In order to define discrete types, we first define a function $\mathsf{path\text{-}to\text{-}hom}$ that converts paths to morphisms for Kan types (`Discrete.path-to-hom`):

$$\begin{aligned}\mathsf{path\text{-}to\text{-}hom}(p) := \lambda(x : 2).\mathsf{com}_A^{z : \mathbb{0}_{\mathbb{I}} \to \mathbb{1}_{\mathbb{I}}}[&x = \mathbb{0}_2 \mapsto p(\mathbb{0}_{\mathbb{I}})\\ &, x = \mathbb{1}_2 \mapsto p(z)](p(\mathbb{0}_{\mathbb{I}}))\end{aligned}$$

Discrete types are then defined by

Definition 3.2.2 ([62, Definition 7.1], `Discrete.isDisc`)**.** A Kan type A is *discrete* when for every $x\ y : A$ the types $\mathsf{Path}_A\,(x, y)$ and $\mathsf{Hom}_A\,(x, y)$ are equivalent along the function $\mathsf{path\text{-}to\text{-}hom}$.

$$\mathsf{isDisc}\ A := \Pi x\ y : A.\mathsf{isEquiv}\ (\mathsf{path\text{-}to\text{-}hom}\ (\lambda\_.A)\ x\ y)$$

In our setting, we define an equivalent (Proposition 3.2.6) property in the style of a cubical filling problem.

Definition 3.2.3. Given a type A, a *discrete filling structure* is a term of the following type:

$$\begin{aligned}\mathsf{discFill}\ A := \quad & \Pi\alpha{:}\Omega_{\mathrm{cof}}. \\ & \Pi t{:}2 \to [\![\alpha]\!]_{\mathsf{cof}} \to A. \\ & \Pi b{:}A[\alpha \mapsto t\ \mathbb{0}_2]. \\ & \Pi i{:}2.A[\alpha \mapsto t\ i, (\mathbb{0}_2 = i) \mapsto b]\end{aligned}$$

In the presence of a witness $\mathsf{discA} : \mathsf{discFill}\ A$, we often write $\mathsf{disc}_A^{z:\mathbb{0}_2 \to i}[\alpha \mapsto t\ z]\ b$ to denote $\mathsf{discA}\ \alpha\ (\lambda z.t\ z)\ b\ i$.

In our proofs, we occasionally use the following interderivable (Lemma 3.2.5) definition when it makes things easier.

$$\begin{aligned}\mathsf{discFill01}\ A := \quad & \Pi\alpha{:}\Omega_{\mathrm{cof}}. \\ & \Pi t{:}2 \to [\![\alpha]\!]_{\mathsf{cof}} \to A. \\ & \Pi b{:}A[\alpha \mapsto t\ \mathbb{0}_2]. \\ & A[\alpha \mapsto t\ \mathbb{1}_2]\end{aligned}$$

Lemma 3.2.4 (`Discrete.discFill-hprop`)**.** *For any Kan type A,* $\mathsf{discFill}\ A$ *is a homotopy proposition.*

Proof. First, consider the type $\Pi b{:}A.\ \mathsf{cfill}\ (\Pi i{:}2.A[\mathbb{0}_2 = i \mapsto b])$. As $\mathsf{cfill}\ X$ is always a homotopy proposition, so is this type. Unfolding the definition of cfill, we see it is the following.

$$\begin{aligned}\Pi b{:}A.\ \mathsf{cfill}\ (\Pi i{:}2.A[\mathbb{0}_2 = i \mapsto b]) = \quad & \Pi b{:}A. \\ & \Pi\alpha{:}\Omega_{\mathrm{cof}}. \\ & \Pi t{:}[\![\alpha]\!]_{\mathsf{cof}} \to \Pi i{:}2.A[\mathbb{0}_2 = i \mapsto b]. \\ & \Pi i{:}2.A[(\mathbb{0}_2 = i) \mapsto b, \alpha \mapsto t\ i]\end{aligned}$$

Rearranging the order of the Πs we can swap α, t and b to be in the same order as in $\mathsf{discFill}\ A$. Then, we can uncurry to combine t and b into a product, and then reorder the product before currying again to move the boundary condition so that it is appended to the type accompanying b, resulting in the type $\mathsf{discFill}\ A$. As all of these transformations are isomorphisms, the type $\mathsf{discFill}\ A$ is isomorphic to $\Pi b{:}A.\ \mathsf{cfill}\ (\Pi i{:}2.A[\mathbb{0}_2 = i \mapsto b])$, and as being a homotopy proposition is preserved by isomorphism, $\mathsf{discFill}\ A$ must be a homotopy proposition. ⊠

Lemma 3.2.5 (`Discrete.discFill-to-discFill01`, `Discrete.discFill01-to-discFill`, [24, Section 4.4])**.** *For any Kan type* A*,* $\mathsf{discFill01}\ A$ *and* $\mathsf{discFill}\ A$ *are interderivable.*

Proof. First, we can provide an instance of $\mathsf{discFill01}\ A$ using $\mathsf{dfillA} : \mathsf{discFill}\ A$ simply by filling to the endpoint $\mathbb{1}_2$.

$$\lambda\alpha\ t\ b.\mathsf{dfillA}\ \alpha\ t\ b\ \mathbb{1}_2 : \mathsf{discFill01}\ A$$

In the other direction, we modify the cofibration to account for the trivial filling and use a connection in the tube we provide to $\mathsf{dfill01A} : \mathsf{discFill01}\ A$ to derive the required instance ot $\mathsf{discFill}\ A$.

$$\lambda\alpha\ t\ b\ i.\mathsf{dfillA01}\ (\alpha \vee \mathbb{0}_2 = i)\ [\alpha \mapsto \lambda x.t\ (i \wedge x), \mathbb{0}_2 = i \mapsto b]\ b : \mathsf{discFill}\ A$$

⊠

Proposition 3.2.6 (`Discrete.isDisc-to-discFill`, `Discrete.discFill-to-isDisc`)**.** *For any Kan type* A*,* $\mathsf{isDisc}\ A$ *and* $\mathsf{discFill}\ A$ *are homotopically equivalent.*

Proof. First, recall from Lemma 3.2.4 that $\mathsf{discFill}\ A$ is a homotopy proposition. As being an equivalence is a homotopy proposition, and quantifying with a Π preserves being a homotopy proposition, $\mathsf{isDisc}\ A$ is one as well. Thus, we only need to show that $\mathsf{isDisc}\ A$ and $\mathsf{discFill}\ A$ are interderivable to conclude they are equivalent.

To begin with, assume we have a witness $\mathsf{discA} : \mathsf{isDisc}\ A$, and let's construct a term of type $\mathsf{discFill}\ A$. To simplify the proof, we will utilize the fact $\mathsf{discFill}\ A$ and $\mathsf{discFill01}\ A$ are interderivable (Lemma 3.2.5) and instead construct a witness to $\mathsf{discFill01}\ A$. As such, let us assume the hypotheses quantified over in the type $\mathsf{discFill01}\ A$: We have an arbitrary cofibration α, a partial tube $\alpha \vdash t : 2 \to A$, and the base we wish to fill $b : A[\alpha \mapsto t\ \mathbb{0}_2]$. Our

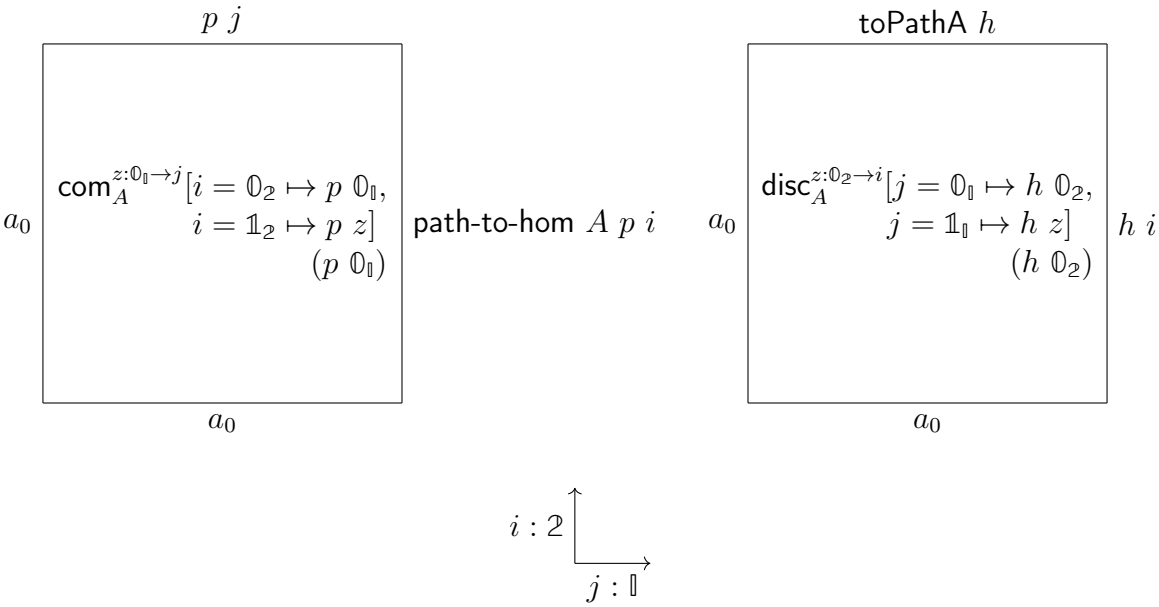
goal is to construct a term of type A such that along the cofibration α it equals $t\ \mathbb{1}_2$. To do this, we first define the partial path $\alpha \vdash t' : \mathsf{Path}_A\,(t\ \mathbb{0}_2)\ (t\ \mathbb{1}_2)$ by applying the inverse to $\mathsf{path\text{-}to\text{-}hom}\ A$ contained in discA to the partial morphism t. We can then use homogenous composition in A to define the desired term by filling from $\mathbb{0}_\mathbb{I}$ to $\mathbb{1}_\mathbb{I}$ along $\alpha \vdash t'$ from b.

$$\lambda\alpha\ t\ b.\mathsf{hcom}_A^{\mathbb{0}_\mathbb{I}\to\mathbb{1}_\mathbb{I}}\ [\alpha \mapsto \mathsf{hom\text{-}to\text{-}path}\ \mathsf{discA}\ t]\ b$$

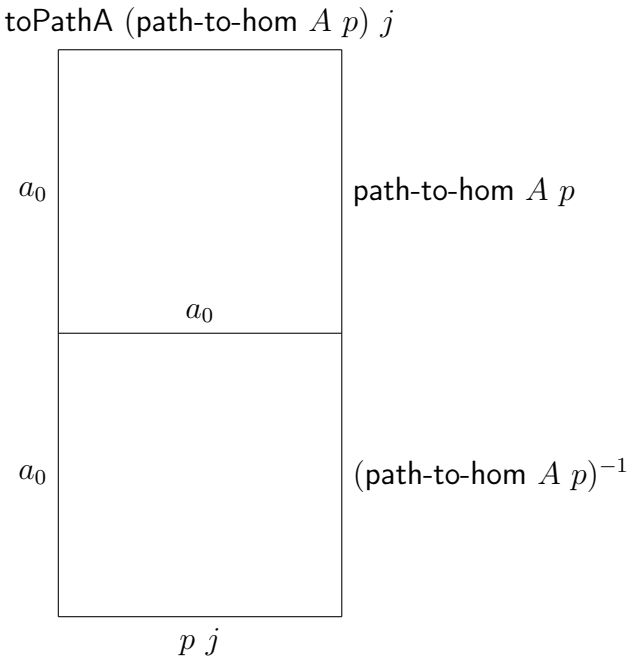
Now assume we have a witness $\mathsf{dfillA} : \mathsf{discFill}\ A$, and let's define a term of type $\mathsf{isDisc}\ A$; in particular, we need to first define a function that turns morphisms in A into paths in A, and then demonstrate said function is an inverse to $\mathsf{path\text{-}to\text{-}hom}\ A$. The inverse $\mathsf{toPathA} : \Pi a_0\ a_1 : A.\mathsf{Hom}_A\,a_0\ a_1 \to \mathsf{Path}_A\,a_0\ a_1$ is straightforward to define:

$$\mathsf{toPathA}\ x\ y\ h := \lambda i : \mathbb{I}.\mathsf{disc}_A^{z:\mathbb{0}_2\to\mathbb{1}_2}\ [i = \mathbb{0}_\mathbb{I} \mapsto h\ \mathbb{0}_2, i = \mathbb{1}_\mathbb{I} \mapsto h\ z]\ (h\ \mathbb{0}_2)$$

To see that $\mathsf{toPathA}$ is an inverse to $\mathsf{path\text{-}to\text{-}hom}\ A$, let us first view the squares used to define each function. To do so, consider $p : \mathsf{Path}_A\,a_0\ a_1$ and $h : \mathsf{Hom}_A\,a_0\ a_1$.

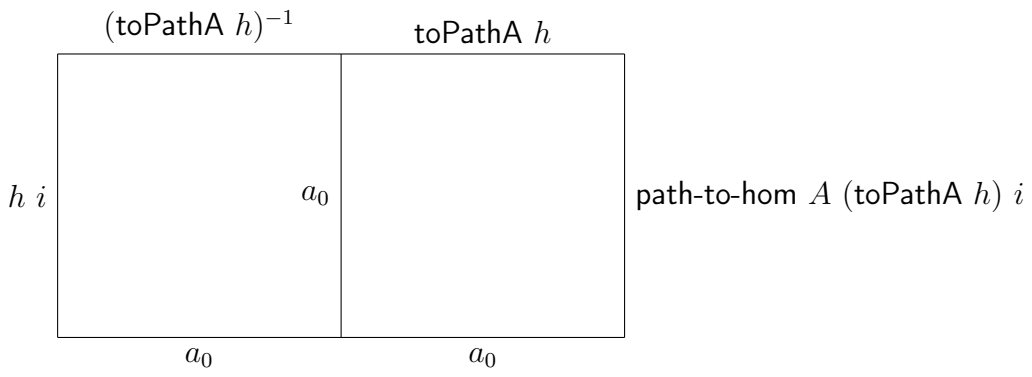


First, let us consider a path p : $\mathsf{Path}_A\, a_0\ a_1$, and construct a path between p and $\mathsf{toPathA}$ ($\mathsf{path\text{-}to\text{-}hom}\ A\ p$). The crux of this proof lies in inverting the $\mathsf{path\text{-}to\text{-}hom}$ square in the directed direction, and then composing it with the $\mathsf{toPathA}$ square instantiated at $\mathsf{path\text{-}to\text{-}hom}\ A\ p$, as shown here.



We can then compose on the right hand side of this square using the path between $\mathsf{path\text{-}to\text{-}hom}\ A\ p \circ (\mathsf{path\text{-}to\text{-}hom}\ A\ p)^{-1}$ and a_1 so that the right-hand side is constantly a_1, and thus we have a morphism from p to $\mathsf{toPathA}$ ($\mathsf{path\text{-}to\text{-}hom}\ A\ p$). Lastly, use discrete filling to turn the morphism into a path, concluding one half of the equivalence.

Now consider a morphism h : $\mathsf{Hom}_A\, a_0\ a_1$, and let's define a path between h and $\mathsf{path\text{-}to\text{-}hom}\ A$ ($\mathsf{toPathA}\ h$). The proof similarly involves inverting one square and pasting the two together. This time, we invert the $\mathsf{toPathA}$ square in the path direction, and attach it to the $\mathsf{path\text{-}to\text{-}hom}$ square instantiated at $\mathsf{toPathA}\ \ \mathsf{h}$.



Composing the top of this square with the path between toPathA $h \circ$ (toPathA h)$^{-1}$ and a_1 results in the desired path between h and path-to-hom A (toPathA h), concluding toPathA is indeed an inverse to path-to-hom A, and thus discFill A and isDisc A are equivalent. ☒

Lemma 3.2.7. *For any Kan type A,* discFill A *is Kan.*

Proof. Recall from Lemma 3.2.4 that discFill A is isomorphic to $\Pi b{:}A.$cfill $(\Pi i{:}\mathbb{2}.A[\mathbb{0}_2 = i \mapsto b])$. This type is clearly Kan as it is built entirely out of Kan types and type formers known to preserve being Kan. as the property of being Kan is preserved by isomorphism, discFill A is Kan as well. ☒

3.2.3 Covariant Discrete Fibrations

Intuitively, a type $B : A \to$ Type is a covariant discrete fibration [62] when it is covariant, in the sense that given a directed morphism $\mathsf{Hom}_A\,(a_0, a_1)$ there is a transport function $B(a_0) \to B(a_1)$ (satisfying a certain universal property), and each fiber $B(a)$ is discrete. In this section, we begin defining the fundemantal components required to ultimately construct a universe $\mathsf{U}_{\mathsf{cov}}$ (which will be completed in Section 3.3), where a type $B : A \to \mathsf{U}_{\mathsf{cov}}$ will be a covariant discrete fibration over A. Because of the directed univalence principle we derive in Chapter 4, $\mathsf{U}_{\mathsf{cov}}$ (modulo a few modifications explained in Chapter 4) itself is a higher-categorical analogue of the category of sets and functions, but where "set" means ∞-

groupoid and function means functor. We could similarly define a universe of contravariant discrete fibrations, but we focus (arbitrarily) on the covariant case for this thesis.

For a type family $A : 2 \to \mathsf{Type}$ to be covariant should mean that there is a "coercion" function $A(\mathbb{0}_2) \to A(\mathbb{1}_2)$. For this to be well-behaved, it is necessary to moreover insist that this function has a universal property relative to A: for each input, its result is related to the input by the relation DHom_A, and in fact the unique such element of $A(\mathbb{1}_2)$. Uniqueness here is up to homotopy (as opposed to strict equality), using the standard notion of a type A being contractible ($\mathsf{iscontr}\ A := \Sigma a : A.\Pi x : A.\mathsf{Path}_A\,(a, x)$)—this style of definition is one reason why the separate notions of morphisms and paths in bisimplicial setting are helpful. For a general type family $A : \Gamma \to \mathsf{Type}$, one asks that any precomposition with a directed path in Γ has such a coercion function:

Definition 3.2.8 ([62, Definition 8.2])**.** Consider a type Γ and type family $A : \Gamma \to \mathsf{Type}$. A is *covariant* if for every morphism $p : \mathsf{Hom}_\Gamma\,(x, y)$ and point $a_0 : A\ x$, there is a unique element $a_1 : A(p\mathbb{1}_2)$ with a dependent path over $A \circ p$ starting at a_0:

$$\begin{aligned}\mathsf{iscov}\ A := \quad & \Pi p{:}2 \to \Gamma.\Pi a_0{:}A\ (p\ \mathbb{0}_2).\\ & \mathsf{iscontr}\ (\Sigma a_1 : A\ (p\ \mathbb{1}_2).\mathsf{DHom}_{A\circ p}\,(a_0, a_1))\end{aligned}$$

While we could use this definition unchanged, in our setting we can give an equivalent definition that will be more convenient, analogous to how we redefined the notion of discrete in Section 3.2.2. First, an equivalent (for Kan types) definition of being contractible is that any partial element can be extended to a complete element [24, 57]:

$$\mathsf{iscontr}'\ A := \Pi\alpha{:}\Omega_{\mathrm{cof}}.\Pi p{:}([\![\alpha]\!]_{\mathsf{cof}} \to A).A[\alpha \mapsto p]$$

The idea is that taking α to be $\top$ gives the center of contraction, and taking α to be $(i = \mathbb{0}_\mathbb{I}) \vee (i = \mathbb{1}_\mathbb{I})$ can be used to create a path between any two elements.

Substituting the partial-extends-to-total definition of contractibility (cfill, Definition 2.3.12) into the definition of covariance and rearranging some quantifiers, we obtain a cubical filling operation for morphism variables:

Definition 3.2.9. A *covariant filling structure* on $A : \Gamma \to \mathsf{Type}$ is a term of type $\mathsf{covFill}_\Gamma\ A$:

$$\begin{aligned}
\mathsf{covFill}^{2}\ (A : 2 \to \mathsf{Type}) :=\ & \Pi\alpha{:}\Omega_{\mathrm{cof}}. \\
& \Pi t{:}(\Pi z{:}2.[\![\alpha]\!]_{\mathsf{cof}} \to A\ z). \\
& \Pi b{:}(A\ \mathbb{0}_2)[\alpha \mapsto t\ \mathbb{0}_2]. \\
& \Pi i{:}2.(A\ i)[\alpha \mapsto t\ i, (\mathbb{0}_2 = i) \mapsto b] \\
\mathsf{covFill}_\Gamma\ (A : \Gamma \to \mathsf{Type}) :=\ & \Pi p{:}2 \to \Gamma.\mathsf{covFill}(A \circ p)
\end{aligned}$$

In the presence of a witness $\mathsf{covA} : \mathsf{covFill}_\Gamma\ A$, we write $\mathsf{cov}_A^{z:\mathbb{0}_2 \to i}\ p\ [\alpha \mapsto t\ z]\ b$ to denote $\mathsf{covA}\ p\ \alpha\ (\lambda z.t\ z)\ b\ i$.

First, we prove some basic facts about the covariant filling problem.

Lemma 3.2.10 (`Covariant.rcov-hprop`)**.** *Given a Kan type A in context Γ, $\mathsf{covFill}_\Gamma\ A$ is a homotopy proposition.*

Proof. This proof is analogous to that showing the discrete filling problem is a homotopy proposition (Lemma 3.2.4).

Consider the type $\Pi p{:}2 \to \Gamma.\Pi b{:}(A \circ p)\ \mathbb{0}_2.\mathsf{cfill}\ (\Pi i{:}2.(A \circ p)\ i[\mathbb{0}_2 = i \mapsto b])$. As contractible fill of a type is always a homotopy proposition, and quantifying by Π preserves a type being a homotopy proposition, this type is a homotopy proposition. By unfolding the definition of cfill and slightly rearranging the order of the Πs and the placement of the boundary condition, we see this type is isomorphic to $\mathsf{covFill}_\Gamma\ A$, and thus $\mathsf{covFill}_\Gamma\ A$ is also a homotopy proposition. ⊠

Theorem 3.2.11 (`Covariant.covFill-internal-equiv`)**.** *Given a Kan type A in context Γ, $\mathsf{covFill}_\Gamma\ A$ is equivalent to* iscov A.

Proof. As shown previously in Lemma 3.2.10, we know that $\mathsf{covFill}_\Gamma\ A$ is isomorphic to the type $\Pi p{:}2 \to \Gamma.\Pi a_0{:}(A \circ p)\ \mathbb{0}_2.\mathsf{cfill}\ (\Pi i{:}2.(A \circ p)\ i[\mathbb{0}_2 = i \mapsto a_0])$. Additionally, the type

of dependent morphisms starting at a_0, $\Pi i{:}2.(A \circ p)\ i[\mathbb{0}_2 = i \mapsto a_0]$, is clearly isomorphic to the dependent sum $\Sigma a_1 : A\ (p\ \mathbb{1}_2).\mathsf{DHom}_{A \circ p}(a_0, a_1)$. Thus, $\mathsf{covFill}_\Gamma\ A$ is isomorphic to $\Pi p{:}2 \to \Gamma.\Pi a_0{:}A\ (p\ \mathbb{0}_2)..\mathsf{cfill}\ (\Sigma a_1 : A\ (p\ \mathbb{1}_2).\mathsf{DHom}_{A \circ p}(a_0, a_1))$. This type only varies from the definition of being covariant (Definition 3.2.8) in that it uses cfill instead of $\mathsf{iscontr}$, but as we know the two are equivalent (Lemma 2.3.14), $\mathsf{covFill}_\Gamma\ A$ is equivalent to $\mathsf{iscov}\ A$. ☒

A proof of the above theorem in the bisimplicial setting is contained in [62, Proposition 8.4].

Lemma 3.2.12 (`Covariant-is-Fibrant.com-covFillI`)**.** *Given a Kan type A in context Γ, $\mathsf{covFill}_\Gamma\ A$ is Kan.*

Proof. As in Lemma 3.2.7, first note that $\mathsf{covFill}_\Gamma\ A$ is isomorphic to $\Pi p{:}2 \to \Gamma.\Pi a_0{:}(A \circ p)\ \mathbb{0}_2.\mathsf{cfill}\ (\Pi i{:}2.(A \circ p)\ i[\mathbb{0}_2 = i \mapsto a_0])$. This type is built out of type-formers that preserve the property of being Kan, and thus is Kan itself. as being Kan is preserved by isomorphisms, $\mathsf{covFill}_\Gamma\ A$ must be Kan as well. ☒

An advantage of this formulation is that many types can be shown to be covariant by a nearly identical construction to their Kan composition structure. For example, because 2 has connections, we can apply the filling-from-composition lemma [24], which states that to construct the above, it suffices to define

$$\begin{aligned}\mathsf{covFill}^{2}_{\mathbb{1}_2}\ (A) :=\ & \Pi\alpha{:}\Omega_{\mathrm{cof}}.\\ & \Pi t{:}(\Pi z{:}2.[\![\alpha]\!]_{\mathsf{cof}} \to A\ z).\\ & \Pi b{:}(A\ \mathbb{0}_2)[\alpha \mapsto t\ \mathbb{0}_2].\\ & (A\ \mathbb{1}_2)[\alpha \mapsto t\ \mathbb{1}_2]\end{aligned}$$

This is exactly CCHM Kan composition [24] but stated for the directed interval, rather than the path interval—thus, we can prove that the covariant types are closed under certain type constructors by the same argument used to show that they are Kan in [24]. This reformulation is also sometimes simpler to prove because it requires constructing one element of A with a specified boundary, as opposed to the two elements of A that constitute a center of contractibility and a path to the center.

3.2.4 Segal Types

We now introduce the first class of well-behaved types we will discuss with nontrivial directed structure. In bisimplicial or bicubical directed type theory, all types have a notion of directed morphism, along with identity morphisms on each object, given by constant functions $(\lambda\_.a) : \mathsf{Hom}_A\,(a, a)$. However, not all types represent categories, in the sense that directed morphisms may not compose. The *Segal condition* is a type describing when a type A does represent a category, by a direct translation of "every compatible pair of morphisms has a unique composition morphism":

First, the triangle is represented by the type $\Sigma(i, j) : 2 \times 2.j \leq i$, recalling that $j \leq i$ is encoded in our setting as $j = j \sqcap i$. The second shape we need for the Segal condition is a pair of compatible arrows. This is given as a subshape of the commuting triangle Δ^2 containing the two contiguous morphisms on its boundary, but not the third or the inside of the triangle. This is often referred to as a *horn* of Δ^2, and is represented by the formula

$$\begin{aligned}&\Lambda^2_1 : \Delta^2 \to \Omega_{\mathrm{cof}}\\ &\Lambda^2_1\ (i, j) := (j = \mathbb{0}_2 \vee i = \mathbb{1}_2, \_)\end{aligned}$$

Then, one way to state the Segal condition is [62]:

$$\begin{aligned}\mathsf{isSegal'}\ A := \quad & \Pi a\ b\ c{:}A.\\ & \Pi p{:}\mathsf{Hom}_A\,(a, b).\\ & \Pi q{:}\mathsf{Hom}_A\,(b, c).\\ & \mathsf{iscontr}(\Pi(i, j){:}\Delta^2.A[j = \mathbb{0}_2 \mapsto p\ i\\ & \qquad\qquad\qquad\qquad , i = \mathbb{1}_2 \mapsto q\ j])\end{aligned}$$

i.e., "every pair of compatible morphisms has a unique composite."

Together a, b, c, p, q constitute a horn in A, so this can be compacted to

Definition 3.2.13 ([62, Definition 5.1])**.** A type A is a *Segal type* if the following condition holds.

$$\begin{aligned}\mathsf{isSegal}\ A := \quad & \Pi a : (\Pi x : \Delta^2.[\![\Lambda^2_1\ x]\!]_{\mathsf{cof}} \to A).\\ & \mathsf{iscontr}(\Pi x : \Delta^2.A[\Lambda^2_1(x) \mapsto a\ x])\end{aligned}$$

As with our other definitions, we show that $\mathsf{isSegal}$ is equivalent (for Kan types) to the following cubical filling problem style definition, which we obtain by using the "any partial element can be extended to a total one" definition of contractibility (cfill, Definition 2.3.12). The analogous definition is also defined and proven equivalent in the bisimplicial setting in Riehl-Shulman [62, Proposition 5.20].

Definition 3.2.14. given a type A, a *Segal filling structure* is a term of the following type:

$$\begin{aligned}\mathsf{SegalFill}\ A :=\ & \Pi\alpha{:}\Omega_{\mathsf{cof}}.\\ & \Pi t{:}(\Pi x : \Delta^2.[\![\alpha]\!]_{\mathsf{cof}} \to A).\\ & \Pi b{:}(\Pi x : \Delta^2.[\![\Lambda^2_1\ x]\!]_{\mathsf{cof}} \to A[\alpha \mapsto t\ x]).\\ & \Pi x{:}\Delta^2.A[\alpha \mapsto t\ x, \Lambda^2_1\ x \mapsto b\ x]\end{aligned}$$

We also show that $\mathsf{SegalFill}\ A$ is Kan and a homotopy proposition, so it could be taken to be the definition of being a category even in a type theory where all types are fibrant.

Lemma 3.2.15. *Given a Kan type A, $\mathsf{SegalFill}\ A$ is a homotopy proposition.*

Proof. The proof is analogous to those showing the other filling problems are homotopy propositions (Lemmas 3.2.4 and 3.2.10), given by restructuring the definition to the isomorphic type

$$\Pi b{:}(\Pi x{:}\Delta^2.[\![\Lambda^2_1\ x]\!]_{\mathsf{cof}} \to A).\mathsf{cfill}\ (\Pi x{:}\Delta^2.A\ [[\![\Lambda^2_1\ x]\!]_{\mathsf{cof}} \mapsto b\ x])$$

that has the structure of Π-quantification followed by a contractible fill type, which is always a homotopy proposition. ⊠

Lemma 3.2.16. *Given a Kan type A, $\mathsf{SegalFill}\ A$ is Kan.*

Proof. As $\Pi b{:}(\Pi x{:}\Delta^2.[\![\Lambda^2_1\ x]\!]_{\mathsf{cof}} \to A).\mathsf{cfill}\ (\Pi x{:}\Delta^2.A\ [[\![\Lambda^2_1\ x]\!]_{\mathsf{cof}} \mapsto b\ x])$ is Kan and isomorphic to $\mathsf{SegalFill}\ A$, $\mathsf{SegalFill}\ A$ is Kan. ⊠

3.2.5 Inner Fibrations

Inner fibrations are a generalization of Segal types in the same way that Kan fibrations generalize types with an hcom structure and the way that covariant fibrations generalize

discrete types; in particular, the fibers of an inner fibration are Segal types. Unfortunately, this notion of fibration has no analogue in 1-category theory, but besides naturally occurring inside of the proofs showing various dependent constructions are Segal, it also is a part of the definition of two very important classes of fibrations: Cartesian and coCartesian fibrations. While these two notions of fibrations are not explored in this thesis, they capture the idea of families of ∞-categories that depend contravariant or covariantly (respectively) on their context; in particular, one can thing of coCartesian fibrations as generalizing our notion of covariant fibrations such that they are families of ∞-categories that depend on their contact covariantly, and are not limited to just being families of ∞-groupoids.

Definition 3.2.17 ([62, Definition 5.19])**.** Consider a type Γ and type family $A : \Gamma \to \mathsf{Type}$. A is an *inner fibration* if for every triangle $p : \Delta^2 \to \Gamma$ and every inner horn $\Pi x : \Delta^2.[\![\Lambda^2_1\ x]\!]_{\mathsf{cof}} \to A\ (p\ x)$, there is a unique horn filler in A:

$$\begin{aligned}\mathsf{isinner}\ A := \quad & \Pi p{:}\Delta^2 \to \Gamma. \\ & \Pi a_\Lambda{:}(\Pi x : \Delta^2.[\![\Lambda^2_1\ x]\!]_{\mathsf{cof}} \to A\ (p\ x)). \\ & \mathsf{iscontr}\ \ (\Pi x : \Delta^2.A\ (p\ x)\ [[\![\Lambda^2_1\ x]\!]_{\mathsf{cof}} \mapsto a_\Lambda\ x])\end{aligned}$$

Definition 3.2.18. An *inner filling structure* on $A : \Gamma \to \mathsf{Type}$ is a term of type $\mathsf{innerFill}_\Gamma\ A$:

$$\begin{aligned}\mathsf{innerFill}^\Delta\ (A : \Delta^2 \to \mathsf{Type}) := \quad & \Pi\alpha{:}\Omega_{\mathrm{cof}}. \\ & \Pi t{:}(\Pi z{:}\Delta^2.[\![\alpha]\!]_{\mathsf{cof}} \to A\ z). \\ & \Pi b{:}(\Pi z{:}\Delta^2.[\![\Lambda^2_1\ z]\!]_{\mathsf{cof}} \to A\ z[\alpha \mapsto t\ z]). \\ & \Pi z{:}\Delta^2.(A\ z)[\alpha \mapsto t\ z, [\![\Lambda^2_1\ z]\!]_{\mathsf{cof}} \mapsto b\ z] \\ \mathsf{innerFill}_\Gamma\ (A : \Gamma \to \mathsf{Type}) := \quad & \Pi p{:}\Delta^2 \to \Gamma.\mathsf{hasInner}(A \circ p)\end{aligned}$$

Lemma 3.2.19. *Given a Kan type A in context Γ, $\mathsf{innerFill}_\Gamma\ A$ is a homotopy proposition.*

Proof. Again analogous to showing the other filling problems are homotopy propositions (Lemmas 3.2.4, 3.2.10 and 3.2.15), given by restructuring the definition to the isomorphic type

$$\Pi p{:}\Delta^2 \to \Gamma.\Pi b{:}(\Pi x{:}\Delta^2.[\![\Lambda^2_1\ x]\!]_{\mathsf{cof}} \to A\ (p\ x)).\mathsf{cfill}\ (\Pi x{:}\Delta^2.A\ (p\ x)\ \left[[\![\Lambda^2_1\ x]\!]_{\mathsf{cof}} \mapsto b\ x\right])$$

that has the structure of Π-quantification followed by a contractible fill type, which is always a homotopy proposition. ⊠

Lemma 3.2.20. *Given a Kan type A in context Γ,* $\mathsf{innerFill}_\Gamma\ A$ *is Kan.*

Proof. As $\mathsf{innerFill}_\Gamma\ A$ is isomorphic to

$$\Pi p{:}\Delta^2 \to \Gamma.\Pi b{:}(\Pi x{:}\Delta^2.[\![\Lambda^2_1\ x]\!]_{\mathsf{cof}} \to A\ (p\ x)).\mathsf{cfill}\ (\Pi x{:}\Delta^2.A\ (p\ x)\ [[\![\Lambda^2_1\ x]\!]_{\mathsf{cof}} \mapsto b\ x]),$$

a Kan type, $\mathsf{innerFill}_\Gamma\ A$ is Kan as well. ⊠

3.3 The Universe $\mathsf{U}_{\mathsf{cov}}$

We construct the universe $\mathsf{U}_{\mathsf{cov}}$ following the LOPS construction [47]: we work within Agda using an idempotent comonadic modality implemented by Andrea Vezzosi[1] to formalize constructions involving the universe. We write $\mathsf{U}_{\mathsf{Kan}}$ for the universe of Kan types constructed in [7]. We build the universe $\mathsf{U}_{\mathsf{cov}}$ of those Kan types that are covariant discrete fibrations ($\mathsf{covFill}$, Definition 3.2.9) using [47, Theorem 5.2]. An important detail to note is that in order to use the LOPS construction to restrict the universe $\mathsf{U}_{\mathsf{Kan}}$, we must minimally modify the type of $\mathsf{covFill}$ such that it classifies type families landing in $\mathsf{U}_{\mathsf{Kan}}$ instead of the ambient universe of all types, with the predicate itself also landing in $\mathsf{U}_{\mathsf{Kan}}$; in other words, for $\Gamma : \mathsf{Type}$, $\mathsf{covFill}_\Gamma : (\Gamma \to \mathsf{U}_{\mathsf{Kan}}) \to \mathsf{U}_{\mathsf{Kan}}$.

Construction 3.3.1. The universe $\mathsf{U}_{\mathsf{cov}}$ that classifies the inner fibrations is definable constructively.

Proof. The assumptions required by the LOPS construction are that (1) that the interval 2 used in the definition of $\mathsf{covFill}^2$ is *tiny*, i.e. exponentiation by the interval has a right adjoint (which we assumed in Figure 3.3); (2) that $\bot$ is a cofibration (an assumption from [7]); and (3) that $\mathsf{covFill}$ classifies Kan types and is itself Kan (Lemma 3.2.12). Note that

[1] This modality has been integrated into Agda as of version 2.6.1.

the covariant predicate would be immediately Kan if we had used the original [62] definition in terms of Π, Σ, path and hom types (all of which are Kan). Given all of the assumptions of the construction are satisfied, we can define the type $\mathsf{U}_{\mathsf{cov}}$ which has decoding function $\mathsf{El} : \mathsf{U}_{\mathsf{cov}} \to \mathsf{U}_{\mathsf{Kan}}$ along with an introduction rule saying that a "crisp" [47] (closed) term of type $\Gamma \to \mathsf{U}_{\mathsf{Kan}}$ has a code in the universe for every covariant discrete fibration structure on it. ⊠

From this construction of the universe, $\mathsf{El} \circ A$ is covariant for any $A : \Gamma \to \mathsf{U}_{\mathsf{cov}}$—any type family in $\mathsf{U}_{\mathsf{cov}}$ has a transport function lifting directed morphisms in its domain to morphisms in $\mathsf{U}_{\mathsf{cov}}$—with the definition of $\mathsf{covFill}(\mathsf{El} \circ A)$ determined by the data in a code. Note that the modal restrictions on universes concern its introduction rule, but the type $\mathsf{U}_{\mathsf{cov}}$ and its elimination forms are unrestricted.

We have checked that this definition of $\mathsf{U}_{\mathsf{cov}}$ is useful by proving it is closed under some example type constructors:

Theorem 3.3.2 (`universe.{Sigma,Pi,Path,Hom}.agda`)**.**
There are codes

- $\Sigma_{\mathsf{cov}} : \Pi(A : \mathsf{U}_{\mathsf{cov}}).\Pi(B : \mathsf{El}A \to \mathsf{U}_{\mathsf{cov}}).\mathsf{U}_{\mathsf{cov}}$
- $\Pi_{\mathsf{cov}} : \Pi(A :^{\flat} \mathsf{U}_{\mathsf{Kan}}).\Pi(B : A \to \mathsf{U}_{\mathsf{cov}}).\mathsf{U}_{\mathsf{cov}}$
- $\mathsf{dpath}_{\mathsf{cov}} : \Pi(A : \mathbb{I} \to \mathsf{U}_{\mathsf{cov}}).\mathsf{El}A\ \mathbb{0}_{\mathbb{I}} \to \mathsf{El}A\ \mathbb{1}_{\mathbb{I}} \to \mathsf{U}_{\mathsf{cov}}$
- $\mathsf{dhom}_{\mathsf{cov}} : \Pi(A : \mathbb{2} \to \mathsf{U}_{\mathsf{cov}}).\mathsf{El}A\ \mathbb{0}_{\mathbb{2}} \to \mathsf{El}A\ \mathbb{1}_{\mathbb{2}} \to \mathsf{U}_{\mathsf{cov}}$

that decode to Σ, Π, DPath *and* DHom *types respectively.*

Proof. To show $\mathsf{U}_{\mathsf{cov}}$ is closed under these type formers, we construct generic witnesses to the filling problem covFill for each.

Σ-types Given a type $A : \mathsf{U}_{\mathsf{cov}}$ and $B : [\![A]\!]_{\mathsf{cov}} \to \mathsf{U}_{\mathsf{cov}}$, we construct a filler of type $\mathsf{covFill}_\Gamma\ (\Sigma x : [\![A]\!]_{\mathsf{cov}}.[\![B\ x]\!]_{\mathsf{cov}})$ in the topos logic as follows. For any morphism $r : 2 \to \Gamma$, cofibration α, tube t and base b, we first define the filling for the first component as the term a.

$$a := \mathsf{cov}_{\Gamma.A}\ p\ [\alpha \mapsto (\mathsf{fst} \circ t)\ r]\ ((\mathsf{fst} \circ b)\ r)$$

We then then use a to define the solution for the entire Σ-type as shown here.

$$\lambda p\ \alpha\ t\ b\ r.(a, \mathsf{cov}_{\Gamma,A.B}\ (\lambda x.(p\ x, a\ x))\ [\alpha \mapsto (\mathsf{snd} \circ t)\ r]\ ((\mathsf{snd} \circ b)\ r))$$

Unsurprisingly, to fill in a pair, one simply just fills in each component directly (taking account of the fact the second component depends on the first).

Π-types Given any flat type $A :^\flat \mathsf{U}_{\mathsf{Kan}}$ and $B : [\![A]\!]_{\mathsf{Kan}} \to \mathsf{U}_{\mathsf{cov}}$, we can always construct a filler of type $\mathsf{covFill}_\Gamma\ (\Pi x : [\![A]\!]_{\mathsf{cov}}.[\![B\ x]\!]_{\mathsf{cov}})$ in the topos logic as shown below.

$$\lambda p\ \alpha\ t\ b\ r\ a.\mathsf{cov}_{\Gamma.Ba}\ p[\alpha \mapsto t\ r\ a]\ (b\ r\ a)$$

As the type A is flat, it does not depend on any path structure and thus there is no need to do any filling in A; therefore one can simply fill these functions by applying their argument and using the filler for B instantiated at the argument.

dependent path types Consider $A : \mathbb{I} \to \mathsf{U}_{\mathsf{cov}}$, along with two terms $a_0 : [\![A\ \mathbb{0}_{\mathbb{I}}]\!]_{\mathsf{cov}}$ and $a_1 : [\![A\ \mathbb{1}_{\mathbb{I}}]\!]_{\mathsf{cov}}$. We wish to construct a solution to the filling problem $\mathsf{covFill}_\Gamma\ (\mathsf{DPath}_A\ (a_0,\ a_1))$. As with the other filling problems, the solution here is to just fill in A and maintain the additional boundary constraints by adding them to the cofibration.

$$\lambda p\ \alpha\ t\ b\ r.(\lambda i.\mathsf{cov}_{\Gamma.A}\ p\ [\alpha \mapsto t\ r\ i, i = \mathbb{0}_{\mathbb{I}} \mapsto a_0\ (p\ r), i = \mathbb{1}_{\mathbb{I}} \mapsto a_1\ (p\ r)]\ (b\ r\ i))$$

dependent hom types Dependent morphism types are handles analogously to dependent path types. For $A : 2 \to \mathsf{U}_{\mathsf{cov}}$ and endpoints a_0 and a_1, the filler is constructed as follows.

$$\lambda p\ \alpha\ t\ b\ r.(\lambda i.\mathsf{cov}_{\Gamma.A}\ p\ [\alpha \mapsto t\ r\ i, i = \mathbb{0}_2 \mapsto a_0\ (p\ r), i = \mathbb{1}_2 \mapsto a_1\ (p\ r)]\ (b\ r\ i))$$

⊠

This shows that $\mathsf{U}_{\mathsf{cov}}$ is closed under Σ types, Π types *with a constant domain* (because the domain is a contravariant and not covariant position; this is enforced by the modal restriction to a crisp A), DPath_A, and DHom_A types. While in general $\mathsf{DHom}_A\,(-, a)$ should be contravariant rather than covariant, if A is in $\mathsf{U}_{\mathsf{cov}}$ it is itself discrete, so its morphisms are equivalent to paths and thus invertible. The full proof is in the above files in the formalization; because of our reformulation of covariance as analogous to CCHM Kan filling, the proofs for Σ, DPath and DHom are analogous to the arguments that Σ and DPath are Kan in [24], while the argument for Π with a constant domain is analogous to the argument that Π has homogeneous composition (which does not use Kan filling in the domain).

3.3.1 $\mathsf{U}_{\mathsf{cov}}$ is Path Univalent

The universe $\mathsf{U}_{\mathsf{cov}}$ should enjoy two kinds of univalence: First, like universes in ordinary homotopy type theory, any path in $\mathsf{U}_{\mathsf{cov}}$ should be the same as an equivalence. Second, any directed morphism should be the same as a function. We write $[\![-]\!]_{\mathsf{cov}}$ for the composite of the two El functions $\mathsf{U}_{\mathsf{cov}} \to \mathsf{U}_{\mathsf{Kan}} \to \mathsf{Type}$. For both kinds of univalence, we use glue types [24], which are formed from a cofibration α, a partial type $T : \alpha \to \mathsf{Type}$, a type $B : \mathsf{Type}$, and a partial function $f : \Pi p : \alpha.T\ p \to B$. The type $\mathsf{Glue}[\alpha \mapsto (T, f)]B$ is isomorphic to $\Sigma t : ([\![\alpha]\!]_{\mathsf{cof}} \to T).B[\alpha \mapsto f\ t]$, but when α is satisfied, $\mathsf{Glue}[\alpha \mapsto (T, f)]B$ is strictly equal to T.

We consider path-univalence first, as its proof is straightforward from previous work. The main lemma required for the proof is that the universe $\mathsf{U}_{\mathsf{cov}}$ is closed under glue types:

Lemma 3.3.3 (`universe.Glue-Equiv-Covariant.GlueUCov`)**.** *Given any* $\alpha : \Omega_{cof}$*,* $T : [\![\alpha]\!]_{\mathsf{cof}} \to \mathsf{U}_{\mathsf{cov}}$*,* $B : \mathsf{U}_{\mathsf{cov}}$*, and* $f : [\![\alpha]\!]_{\mathsf{cof}} \to [\![T]\!]_{\mathsf{cov}} \to [\![B]\!]_{\mathsf{cov}}$ *where* f *is an equivalence, the type* $\mathsf{Glue}[\alpha \mapsto ([\![T]\!]_{\mathsf{cov}}, f)][\![B]\!]_{\mathsf{cov}}$ *is in* $\mathsf{U}_{\mathsf{cov}}$*.*

Proof. To conclude this, we need to provide an instance to the covariant filling problem for the type $\mathsf{Glue}[\alpha \mapsto ([\![T]\!]_{\mathsf{cov}}, f)][\![B]\!]_{\mathsf{cov}}$. As B is in $\mathsf{U}_{\mathsf{cov}}$, we have a witness to $\mathsf{covFill}_\Gamma\ B$. We also know from [24, Theorem 9] that the map $\mathsf{unglue} : \mathsf{Glue}[\alpha \mapsto ([\![T]\!]_{\mathsf{cov}}, f)][\![B]\!]_{\mathsf{cov}} \to B$ is an equivalence. Thus, using univalence in $\mathsf{U}_{\mathsf{Kan}}$, we can convert the unglue equivalence into a path and transport along it in $\mathsf{covFill}_\Gamma$ to derive a solution for the filling problem for the glue type from the one we have for B. ⊠

Glue types have univalence as the special case of gluing with an equivalence on one side: $\mathsf{Glue}[x = \mathbb{0}_{\mathbb{I}} \mapsto (A, e), (x = 1) \mapsto (B, \mathsf{id}_B)](b)$.

Theorem 3.3.4 (`DirUnivalence.ua`)**.** *For any two types* A *and* B *in* $\mathsf{U}_{\mathsf{cov}}$*, we have the following equivalence.*

$$\mathsf{ua} : \mathsf{Equiv}\ (\mathsf{Equiv}\ [\![A]\!]_{\mathsf{cov}}\ [\![B]\!]_{\mathsf{cov}})\ (\mathsf{Path}_{\mathsf{U}_{\mathsf{cov}}}\ (A, B))$$

Proof. First, as any types in $\mathsf{U}_{\mathsf{cov}}$ also have codes in $\mathsf{U}_{\mathsf{Kan}}$, we know a path $\mathbb{I} \to \mathsf{U}_{\mathsf{cov}}$ is also a path in $\mathsf{U}_{\mathsf{Kan}}$, and thus are able to derive an equivalence from it using univalence in $\mathsf{U}_{\mathsf{Kan}}$.

Given two types A and B in $\mathsf{U}_{\mathsf{cov}}$ and an equivalence between them, we also know from Lemma 3.3.3 that the construction used by univalence in $\mathsf{U}_{\mathsf{Kan}}$ to define a corresponding path from A to B also has a code in $\mathsf{U}_{\mathsf{cov}}$, and thus lifts to a path from A to B in $\mathsf{U}_{\mathsf{cov}}$.

As both the function from equivalences to paths and that from paths to equivalences are given identically as for univalence in $\mathsf{U}_{\mathsf{Kan}}$, we already know they are homotopy inverses, and thus form an equivalence. ⊠

Glue types also imply that $\mathsf{U}_{\mathsf{cov}}$ is itself Kan—given some paths to compose in $\mathsf{U}_{\mathsf{cov}}$, we can project paths in $\mathsf{U}_{\mathsf{Kan}}$, which determine equivalences, which can then be composed using the glue type, which is covariant by above.

Theorem 3.3.5 (`universe.UCov-Com.UCovU`)**.** $\mathsf{U}_{\mathsf{cov}}$ *is Kan.*

Proof. This proof is analogous to the proofs that $\mathsf{U}_{\mathsf{Kan}}$ is Kan in [24, Section 7.1] and [7, Section 2.12]. ⊠

3.4 The Universe $\mathsf{U}_{\mathsf{inner}}$

The Universe $\mathsf{U}_{\mathsf{inner}}$ classifies the inner fibrations. We choose to construct this universe as its fibers are the Segal types—our representation of ∞-categories—and given it is the "dependently typed" version of the filling problem it has better type closure properties; to fill in a dependent type one often needs to, unsurprisingly, fill dependently.

Construction 3.4.1. The universe $\mathsf{U}_{\mathsf{inner}}$ that classifies the inner fibrations is definable constructively.

Proof. As with our definition of $\mathsf{U}_{\mathsf{cov}}$, we use the LOPS construction [47] to define the universe $\mathsf{U}_{\mathsf{inner}}$ as a restriction of the universe $\mathsf{U}_{\mathsf{Kan}}$. The primary distinction in this definition is that the predicate defining the inner filling structure (Definition 3.2.18) is abstracted over the directed triangle Δ^2 instead of an interval. That being said, Δ^2 is still tiny (Lemma 3.1.5) as it is an idempotent subobject of a representable. Also note that by Lemma 3.2.20, we can modify the predicate as required such that it classifies Kan fibrations and lands in $\mathsf{U}_{\mathsf{Kan}}$. Putting this all together the LOPS construction functions as required. ⊠

We also show $\mathsf{U}_{\mathsf{inner}}$ is closed under a number of standard type formers:

Theorem 3.4.2. *There are codes*

- $\Sigma_{\mathsf{inner}} : \Pi(A : \mathsf{U}_{\mathsf{inner}}).\Pi(B : \mathsf{El}A \to \mathsf{U}_{\mathsf{inner}}).\mathsf{U}_{\mathsf{inner}}$
- $\Pi_{\mathsf{inner}} : \Pi(A :^{\flat} \mathsf{U}_{\mathsf{Kan}}).\Pi(B : A \to \mathsf{U}_{\mathsf{inner}}).\mathsf{U}_{\mathsf{inner}}$
- $\mathsf{dpath}_{\mathsf{inner}} : \Pi(A : \mathbb{I} \to \mathsf{U}_{\mathsf{inner}}).\mathsf{El}A\ \mathbb{0}_{\mathbb{I}} \to \mathsf{El}A\ \mathbb{1}_{\mathbb{I}} \to \mathsf{U}_{\mathsf{inner}}$
- $\mathsf{dhom}_{\mathsf{inner}} : \Pi(A : \mathbb{2} \to \mathsf{U}_{\mathsf{inner}}).\mathsf{El}A\ \mathbb{0}_{\mathbb{2}} \to \mathsf{El}A\ \mathbb{1}_{\mathbb{2}} \to \mathsf{U}_{\mathsf{inner}}$

that decode to Σ, Π, DPath *and* DHom *types respectively.*

Proof. To show $\mathsf{U}_{\mathsf{inner}}$ is closed under these type formers, we construct generic witnesses to the filling problem $\mathsf{innerFill}$ for each.

Σ-types Given a type $A : \mathsf{U}_{\mathsf{inner}}$ and $B : [\![A]\!]_{\mathsf{inner}} \to \mathsf{U}_{\mathsf{inner}}$, we construct a filler of type $\mathsf{innerFill}_\Gamma\ (\Sigma x : [\![A]\!]_{\mathsf{inner}}.[\![B\ x]\!]_{\mathsf{inner}})$ in the topos logic as follows. For any triangle $r : \Delta^2 \to \Gamma$, cofibration α, tube t and base b, we first define the filling for the first component as the term a.

$$a := \mathsf{inner}_{\Gamma.A}\ p\ [\alpha \mapsto (\mathsf{fst} \circ t)\ r]\ ((\mathsf{fst} \circ b)\ r)$$

We then then use a to define the solution for the entire Σ-type as shown here.

$$\lambda p\ \alpha\ t\ b\ r.(a, \mathsf{inner}_{\Gamma,A.B}\ (\lambda x.(p\ x, a\ x))\ [\alpha \mapsto (\mathsf{snd} \circ t)\ r]\ ((\mathsf{snd} \circ b)\ r))$$

Unsurprisingly, to fill in a pair, one simply just fills in each component directly (taking account of the fact the second component depends on the first).

Π-types Given any flat type $A :^\flat \mathsf{U}_{\mathsf{Kan}}$ and $B : [\![A]\!]_{\mathsf{Kan}} \to \mathsf{U}_{\mathsf{inner}}$, we can always construct a filler of type $\mathsf{innerFill}_\Gamma\ (\Pi x : [\![A]\!]_{\mathsf{inner}}.[\![B\ x]\!]_{\mathsf{inner}})$ in the topos logic as shown below.

$$\lambda p\ \alpha\ t\ b\ r\ a.\mathsf{inner}_{\Gamma.Ba}\ p[\alpha \mapsto t\ r\ a]\ (b\ r\ a)$$

As the type A is flat, it does not depend on any path structure and thus there is no need to do any filling in A; therefore one can simply fill these functions by applying their argument and using the filler for B instantiated at the argument.

dependent path types Consider $A : \mathbb{I} \to \mathsf{U}_{\mathsf{inner}}$, along with two terms $a_0 : [\![A\ \mathbb{0}_{\mathbb{I}}]\!]_{\mathsf{inner}}$ and $a_1 : [\![A\ \mathbb{1}_{\mathbb{I}}]\!]_{\mathsf{inner}}$. We wish to construct a solution to the filling problem $\mathsf{innerFill}_\Gamma\ (\mathsf{DPath}_A\ (a_0,\ a_1))$. As with the other filling problems, the solution here is to just fill in A and maintain the

additional boundary constraints by adding them to the cofibration.

$$\lambda p\ \alpha\ t\ b\ r.(\lambda i.\mathsf{inner}_{\Gamma.A}\ p\ [\alpha \mapsto t\ r\ i, i = \mathbb{0}_{\mathbb{I}} \mapsto a_0\ (p\ r), i = \mathbb{1}_{\mathbb{I}} \mapsto a_1\ (p\ r)]\ (b\ r\ i))$$

dependent hom types Dependent morphism types are handles analogously to dependent path types. For $A : \mathbb{2} \to \mathsf{U}_{\mathsf{inner}}$ and endpoints a_0 and a_1, the filler is constructed as follows.

$$\lambda p\ \alpha\ t\ b\ r.(\lambda i.\mathsf{inner}_{\Gamma.A}\ p\ [\alpha \mapsto t\ r\ i, i = \mathbb{0}_{\mathbb{2}} \mapsto a_0\ (p\ r), i = \mathbb{1}_{\mathbb{2}} \mapsto a_1\ (p\ r)]\ (b\ r\ i))$$

⊠

3.4.1 $\mathsf{U}_{\mathsf{inner}}$ is Path Univalent

As a subuniverse of $\mathsf{U}_{\mathsf{Kan}}$, $\mathsf{U}_{\mathsf{inner}}$ also enjoys path univalence in the same way $\mathsf{U}_{\mathsf{cov}}$ does.

Lemma 3.4.3. *In any context* Γ*, given any* $\alpha : \Omega_{cof}$*,* $T : [\![\alpha]\!]_{\mathsf{cof}} \to \mathsf{U}_{\mathsf{inner}}$*,* $B : \mathsf{U}_{\mathsf{inner}}$*, and* $f : [\![\alpha]\!]_{\mathsf{cof}} \to [\![T]\!]_{\mathsf{inner}} \to [\![B]\!]_{\mathsf{inner}}$ *where* f *is an equivalence, the type* $\mathsf{Glue}[\alpha \mapsto ([\![T]\!]_{\mathsf{inner}}, f)][\![B]\!]_{\mathsf{inner}}$ *is in* $\mathsf{U}_{\mathsf{inner}}$*.*

Proof. Here we need to provide an instance to the inner filling problem for the type $\mathsf{Glue}[\alpha \mapsto ([\![T]\!]_{\mathsf{cov}}, f)][\![B]\!]_{\mathsf{cov}}$. As B is already in $\mathsf{U}_{\mathsf{inner}}$, we have a witness to $\mathsf{innerFill}_\Gamma\ B$. As we did in Lemma 3.3.3, we use combine the fact from [24, Theorem 9] that the map $\mathsf{unglue} : \mathsf{Glue}[\alpha \mapsto ([\![T]\!]_{\mathsf{cov}}, f)][\![B]\!]_{\mathsf{cov}} \to B$ is an equivalence with univalence in $\mathsf{U}_{\mathsf{Kan}}$ to transport the witness that B is an inner fibration along $\mathsf{innerFill}_\Gamma$ to construct a witness for the glue type. ⊠

Theorem 3.4.4. *For any two types* A *and* B *in* $\mathsf{U}_{\mathsf{inner}}$*, we have the following equivalence.*

$$\mathsf{ua} : \mathsf{Equiv}\ (\mathsf{Equiv}\ [\![A]\!]_{\mathsf{inner}}\ [\![B]\!]_{\mathsf{inner}})\ (\mathsf{Path}_{\mathsf{U}_{\mathsf{inner}}}\ (A, B))$$

Proof. As $\mathsf{U}_{\mathsf{inner}}$ is closed under glue types for equivalences, the same argument used to construct univalence for $\mathsf{U}_{\mathsf{cov}}$ (Theorem 3.3.4) also applies here. ⊠

Theorem 3.4.5. $\mathsf{U}_{\mathsf{inner}}$ *is Kan.*

Proof. As with $\mathsf{U}_{\mathsf{cov}}$, the closure of $\mathsf{U}_{\mathsf{inner}}$ with respect to glue types for equivalences allows us to use the same reasoning as is done to show that $\mathsf{U}_{\mathsf{Kan}}$ is Kan in [24, Section 7.1] and [7, Section 2.12]. ⊠

Chapter 4

Directed Univalence

Like the previous chapter, this chapter is also joint work with Daniel Licata and is based primarily off of work presented at the 35th Annual ACM/IEEE Symposium on Logic in Computer Science and is published in the corresponding proceedings [83]. Versions of this work have also been presented numerous times both before and after publication including those listed here [79–81].

Having defined the groundwork for bicubical directed type theory in Chapter 3, we now shift to developing the payoff of our setup: constructive directed univalence. In this chapter, we define an equivalence between the types of directed morphisms and the function spaces within the universe of covariant discrete fibrations. We do so in two steps: First, in Section 4.1, we build directed off of the theory and techniques in Chapter 3 to define a retraction from the morphisms onto the function spaces in $\mathsf{U}_{\mathsf{cov}}$ (Lemma 4.1.3); at this point we need to know slightly more about our types to upgrade this reflection to an equivalence, and thus in Section 4.2 we introduce the cobar modality and by restricting the entire type theory to those types that are modal we gain enough traction to complete directed univalence.

4.1 The Directed Univalence Retraction

Here we define the directed univalence reflection of morphisms $\mathsf{Hom}_{\mathsf{U}_{\mathsf{cov}}}\,(A, B)$ onto functions $[\![A]\!]_{\mathsf{cov}} \to [\![B]\!]_{\mathsf{cov}}$.

4.1.1 Morphisms to Functions

Given a morphism in $\mathsf{U}_{\mathsf{cov}}$, part of the definition of $\mathsf{U}_{\mathsf{cov}}$ is a covariant filling structure for $\mathsf{El}_{\mathsf{cov}} : \mathsf{U}_{\mathsf{cov}} \to \mathsf{Type}$, which yields a coercion function along from $\mathbb{0}_2$ to $\mathbb{1}_2$:

$$\begin{aligned}&\mathsf{dcoe} : \Pi A\ B : \mathsf{U}_{\mathsf{cov}}.\mathsf{Hom}_{\mathsf{U}_{\mathsf{cov}}}\,(A, B) \to ([\![A]\!]_{\mathsf{cov}} \to [\![B]\!]_{\mathsf{cov}})\\ &\mathsf{dcoe}\ A\ B\ p := \lambda x : \mathsf{cov}_{\mathsf{El}}\ p\ \bot\ \mathsf{exfalso}\ x\ \mathbb{1}_2\end{aligned}$$

4.1.2 Functions to Morphisms

We turn a function into a morphism using the same glue type used for undirected univalence, but this time without requiring that the function f be an equivalence:

$$\begin{aligned}&\mathsf{duahom}' : \Pi A\ B : \mathsf{Type}.(A \to B) \to \mathsf{Hom}_{\mathsf{Type}}\,(A, B)\\ &\mathsf{duahom}'\ A\ B\ f := \lambda i : 2.\mathsf{Glue}[i = \mathbb{0}_2 \mapsto (A, f)\\ &\qquad\qquad\qquad\qquad , i = \mathbb{1}_2 \mapsto (B, \mathsf{id}_B)]B\end{aligned}$$

In presenting the definition of duahom', we omit the boundary proofs for the sake of clarity.

For this to be a morphism in $\mathsf{U}_{\mathsf{cov}}$, we need to show that it is Kan and covariant. In general, glue types with an arbitrary function (as opposed to an equivalence) are not Kan, but the fact that the cofibration restricts a *directed* interval i gives some additional leverage that is not present in ordinary cubical models.

Lemma 4.1.1 (`universe.FunGlue.FunGlueUKan`)**.** *For every pair of types* $A\ B : \mathsf{U}_{\mathsf{Kan}}$*, function* $f : [\![A]\!]_{\mathsf{Kan}} \to [\![B]\!]_{\mathsf{Kan}}$*, and* $i : \mathbb{2}$*, the type* $\mathsf{duahom}'\ A\ B\ f\ i$ *determines a code in* $\mathsf{U}_{\mathsf{Kan}}$*.*

Proof. We need to show that duahom' is a Kan fibration. Thus, for any path p in $\mathbb{2} \to (\Sigma A : \mathsf{U}_{\mathsf{Kan}}.\Sigma B : \mathsf{U}_{\mathsf{Kan}}.A \to B)$, we construct a solution to the $\mathsf{comFill}^{\mathbb{I}}$ filling problem for the path in Type induced by the composition of p and duahom'. For the sake of clarity, we will only present the proof that this path has a coercion structure (i.e. $\mathsf{comFill}^{\mathbb{I}}$ from $\mathbb{0}_{\mathbb{I}}$ to $\mathbb{1}_{\mathbb{I}}$ over the false cofibration proposition $\bot$), but the full proof is available in the Agda formalization.

Let p_i be the path in $\mathbb{2}$ induced by p, p_A and p_B be the two paths in $\mathsf{U}_{\mathsf{Kan}}$ induced by p, and lastly p_f be the dependent path over $\lambda j.[\![p_A\ j]\!]_{\mathsf{Kan}} \to [\![p_B\ j]\!]_{\mathsf{Kan}}$ contained in p. Now, given any term

$$g_0 : \mathsf{duahom}'\ [\![p_A\ \mathbb{0}_{\mathbb{I}}]\!]_{\mathsf{Kan}}\ [\![p_B\ \mathbb{0}_{\mathbb{I}}]\!]_{\mathsf{Kan}}\ (p_f\ \mathbb{0}_{\mathbb{I}})\ (p_i\ \mathbb{0}_{\mathbb{I}})$$

we must construct a term in

$$\mathsf{duahom}'\ [\![p_A\ \mathbb{1}_{\mathbb{I}}]\!]_{\mathsf{Kan}}\ [\![p_B\ \mathbb{1}_{\mathbb{I}}]\!]_{\mathsf{Kan}}\ (p_f\ \mathbb{1}_{\mathbb{I}})\ (p_i\ \mathbb{1}_{\mathbb{I}})$$

Note that, for any A, B, f, and i, a term in $\mathsf{duahom}'\ A\ B\ f\ i$ is just the pair of a term $a : A$ under the assumption $i = \mathbb{0}_{\mathbb{2}}$, and a term $b : B$ where $f\ a$ is strictly equal to b when $i = \mathbb{0}_{\mathbb{2}}$. Thus, let a_0 and b_0 be the components defining g_0. Let a_1 and b_1 denote the corresponding components we must construct to define a term in $\mathsf{duahom}'\ [\![p_A\ \mathbb{1}_{\mathbb{I}}]\!]_{\mathsf{Kan}}\ [\![p_B\ \mathbb{1}_{\mathbb{I}}]\!]_{\mathsf{Kan}}\ (p_f\ \mathbb{1}_{\mathbb{I}})\ (p_i\ \mathbb{1}_{\mathbb{I}})$.

As p_A is a path in $\mathsf{U}_{\mathsf{Kan}}$ and thus is Kan fibration, we can define a_1 by coercing a_0 along p_A, which we will denote as

$$a_1 := \mathsf{com}^{\mathbb{0}_{\mathbb{I}} \mapsto \mathbb{1}_{\mathbb{I}}} p_A\ a_0$$

We must be able to define a_1 whenever $p_i\ \mathbb{1}_{\mathbb{I}} = \mathbb{0}_{\mathbb{2}}$, but it depends on a_0 which only exists when $p_i\ \mathbb{0}_{\mathbb{I}} = \mathbb{0}_{\mathbb{2}}$. Thus, for this to work, we require the triviality axiom ($\mathbb{I}\mathsf{to2triv}$) that states any function $\mathbb{I} \to \mathbb{2}$ is trivial. In our setting, this means that if $p_i\ \mathbb{1}_{\mathbb{I}} = \mathbb{0}_{\mathbb{2}}$ we know that for

every $j : \mathbb{I}$ it must be the case that $p_i\ j = \mathbb{0}_{\mathbb{2}}$, and therefore $p_i\ \mathbb{0}_{\mathbb{I}} = \mathbb{0}_{\mathbb{2}}$ confirming a_0 exists whenever we need to define a_1.

Next, to define b_1, we analogously fill along p_B, but to ensure it is strictly equal to $f\ \mathbb{1}_{\mathbb{I}}\ a_1$ when $p_i\ \mathbb{1}_{\mathbb{I}} = \mathbb{0}_{\mathbb{2}}$, we must fill along the cofibration proposition $p_i\ \mathbb{1}_{\mathbb{I}} = \mathbb{0}_{\mathbb{2}}$.

$$b_1 := \mathsf{com}^{\mathbb{0}_{\mathbb{I}} \mapsto \mathbb{1}_{\mathbb{I}}} p_B[p_i\ \mathbb{1}_{\mathbb{I}} = \mathbb{0}_{\mathbb{2}} \mapsto \lambda j.p_f\ j\ (\mathsf{com}^{\mathbb{0}_{\mathbb{I}} \mapsto j} p_A\ a_0)]\ b_0$$

Again, b_1 also depends on $\mathbb{I}\mathsf{to}\mathbb{2}\mathsf{triv}$ in order to define the boundary constraint.

The construction for a general composition problem in our formalization is nearly identical to that of the coercion described above. As duahom' is Kan, for every pair of types $A\ B : \mathsf{U}_{\mathsf{Kan}}$, function $f : [\![A]\!]_{\mathsf{Kan}} \to [\![B]\!]_{\mathsf{Kan}}$, and $i : \mathbb{2}$, the type $\mathsf{duahom}'\ A\ B\ f\ i$ determines a code in $\mathsf{U}_{\mathsf{Kan}}$. ⊠

Lemma 4.1.2 (`universe.FunGlue.FunGlueUCov`)**.** *For every pair of types $A\ B : \mathsf{U}_{\mathsf{cov}}$, function $f : [\![A]\!]_{\mathsf{cov}} \to [\![B]\!]_{\mathsf{cov}}$, and $i : \mathbb{2}$, the type $\mathsf{duahom}'\ A\ B\ f\ i$ determines a code in $\mathsf{U}_{\mathsf{cov}}$.*

Proof. As we already know that duahom' is Kan, this amounts to showing it is covariant. The construction of the solution of the covariant filling problem is nearly identical to that of the Kan filling problem. The sole difference is that we use covariant filling in A and B instead of Kan filling, and we have a different justification for why a_0 always exists whenever we need to be able to define a_1.

Let p_i be the morphism in $\mathbb{2}$ induced by the morphism we are considering in the filling problem. We must be able to define a_1 whenever $p_i\ \mathbb{1}_{\mathbb{2}} = \mathbb{0}_{\mathbb{2}}$ using a_0 which only exists when $p_i\ \mathbb{0}_{\mathbb{2}} = \mathbb{0}_{\mathbb{2}}$. In this case, we use the monotonicity axiom ($\mathbb{2}\mathsf{mono}$): if $p_i\ \mathbb{1}_{\mathbb{2}} = \mathbb{0}_{\mathbb{2}}$, we know that for every $j : \mathbb{2}$ it also holds that $p_i\ j = \mathbb{0}_{\mathbb{2}}$, and thus we always have a_0 when it is needed to define a_1.

Thus, for any morphism p in $\mathbb{2} \times (\Sigma A : \mathsf{U}_{\mathsf{cov}}.\Sigma B : \mathsf{U}_{\mathsf{cov}}.A \to B)$, we construct a solution to the $\mathsf{covFill}^{\mathbb{2}}$ filling problem for the path in Type induced by the composition of p and duahom'. For the sake of clarity, we will only present the proof that this path has a coercion

structure (i.e. $\mathsf{covFill}^2$ from $\mathbb{0}_2$ to $\mathbb{1}_2$ over the false cofibration proposition $\bot$), but the full proof is available in the Agda formalization.

Let p_i be the morphism in 2 induced by p, p_A and p_B be the two morphisms in $\mathsf{U_{cov}}$ induced by p, and lastly p_f be the dependent morphism over $\lambda j.[\![p_A\ j]\!]_{\mathsf{cov}} \to [\![p_B\ j]\!]_{\mathsf{cov}}$ contained in p. Now, given any term

$$g_0 : \mathsf{duahom'}\ [\![p_A\ \mathbb{0}_2]\!]_{\mathsf{cov}}\ [\![p_B\ \mathbb{0}_2]\!]_{\mathsf{cov}}\ (p_f\ \mathbb{0}_2)\ (p_i\ \mathbb{0}_2)$$

we must construct a term in $\mathsf{duahom'}\ [\![p_A\ \mathbb{1}_2]\!]_{\mathsf{cov}}\ [\![p_B\ \mathbb{1}_2]\!]_{\mathsf{cov}}\ (p_f\ \mathbb{1}_2)\ (p_i\ \mathbb{1}_2)$. As mentioned in the proof that $\mathsf{duahom'}$ is a Kan fibration, a term in $\mathsf{duahom'}\ A\ B\ f\ i$ is just the pair of a term $a : A$ under the assumption $i = \mathbb{0}_2$, and a term $b : B$ where $f\ a$ is strictly equal to b when $i = \mathbb{0}_2$. Thus, let a_0 and b_0 be the components defining g_0. Let a_1 and b_1 denote the corresponding components we must construct to define a term in $\mathsf{duahom'}\ [\![p_A\ \mathbb{1}_2]\!]_{\mathsf{cov}}\ [\![p_B\ \mathbb{1}_2]\!]_{\mathsf{cov}}\ (p_f\ \mathbb{1}_2)\ (p_i\ \mathbb{1}_2)$.

First, as p_A is a morphism in $\mathsf{U_{cov}}$ and thus is covariant fibration, we can define a_1 by coercing a_0 along p_A, which we will denote as

$$a_1 := \mathsf{cov}^{\mathbb{0}_2 \mapsto \mathbb{1}_2} p_A\ a_0$$

We must be able to define a_1 whenever $p_i\ \mathbb{1}_2 = \mathbb{0}_2$, but it depends on a_0 which only exists when $p_i\ \mathbb{0}_2 = \mathbb{0}_2$. Thus, for this to work, we require the monotonicity axiom that states any function $2 \to 2$ is monotonic. In our setting, this means that if $p_i\ \mathbb{1}_2 = \mathbb{0}_2$ we know that for every $j : 2$ it must be the case that $p_i\ j = \mathbb{0}_2$, and therefore $p_i\ \mathbb{0}_2 = \mathbb{0}_2$ confirming a_0 exists whenever we need to define a_1.

Next, to define b_1, we analogously fill along p_B, but to ensure it is strictly equal to $f\ \mathbb{1}_2\ a_1$ when $p_i\ \mathbb{1}_2 = \mathbb{0}_2$, we must fill along the cofibration proposition $p_i\ \mathbb{1}_2 = \mathbb{0}_2$.

$$b_1 := \mathsf{cov}^{\mathbb{0}_2 \mapsto \mathbb{1}_2} p_B[p_i\ \mathbb{1}_2 = \mathbb{0}_2 \mapsto \lambda j.p_f\ j\ (\mathsf{cov}^{\mathbb{0}_2 \mapsto j} p_A\ a_0)]\ b_0$$

Again, b_1 also depends on monotonicity in order to define the boundary constraint.

While there are more technical details in defining the solution to the full covariant filling problem, the construction is nearly identical to that of the coercion described above. As duahom' is both Kan and covariant, for every pair of types $A\ B : \mathsf{U}_{\mathsf{cov}}$, function $f : [\![A]\!]_{\mathsf{cov}} \to [\![B]\!]_{\mathsf{cov}}$, and $i : 2$, the type $\mathsf{duahom}'\ A\ B\ f\ i$ determines a code in $\mathsf{U}_{\mathsf{cov}}$. ⊠

Since duahom' is both Kan and covariant, it constructs a morphism in $\mathsf{U}_{\mathsf{cov}}$:

$$\mathsf{duahom} : \Pi A\ B : \mathsf{U}_{\mathsf{cov}}.([\![A]\!]_{\mathsf{cov}} \to [\![B]\!]_{\mathsf{cov}}) \to \mathsf{Hom}_{\mathsf{U}_{\mathsf{cov}}}(A, B)$$

4.1.3 Reflection

A calculation with the definition of Lemma 4.1.2 shows that

Lemma 4.1.3 (`DirUnivalence.dua`β)**.** *For every pair types $A\ B : \mathsf{U}_{\mathsf{cov}}$ and every function $f : [\![A]\!]_{\mathsf{cov}} \to [\![B]\!]_{\mathsf{cov}}$, we can construct a path*

$$\mathsf{dua}\beta : \mathsf{Path}_{[\![A]\!]_{\mathsf{cov}} \to [\![B]\!]_{\mathsf{cov}}}(f, \mathsf{dcoe}\ A\ B\ (\mathsf{duahom}\ A\ B\ f))$$

Proof. To define the path, we construct the following function of type $\mathbb{I} \to A \to B$:

$$\begin{aligned}&\lambda i\ a.\mathsf{cov}^{\mathbb{0}_2 \mapsto \mathbb{1}_2}(\mathsf{duahom}\ A\ B\ f)\\ &\quad [i = \mathbb{0}_{\mathbb{I}} \mapsto \lambda j.\mathsf{glue}[j = \mathbb{0}_2 \mapsto a, j = \mathbb{1}_2 \mapsto f\ a]f\ a\\ &\quad , i = \mathbb{1}_{\mathbb{I}} \mapsto \lambda j.\mathsf{cov}^{\mathbb{0}_2 \mapsto j}(\mathsf{duahom}\ A\ B\ f)\ a]\ a\end{aligned}$$

As the function has the correct boundaries at $i = \mathbb{0}_{\mathbb{I}}$ and $i = \mathbb{1}_{\mathbb{I}}$, it is the desired path. ⊠

However, for the other composite, we obtain only a morphism:

Lemma 4.1.4 (`DirUnivalence.dua$\eta$fun`)**.** *For every* $A\ B : \mathsf{U}_{\mathsf{cov}}$*,* $p : \mathsf{Hom}_{\mathsf{U}_{\mathsf{cov}}}(A, B)$*, and* $i : \mathbb{2}$*, we can construct a function (and therefore a* $\mathsf{Hom}_{\mathsf{U}_{\mathsf{cov}}}$ *by Lemma 4.1.2)*

$$\mathsf{dua}\eta\mathsf{fun}\ i : (p\ i) \to (\mathsf{duahom}\ A\ B\ (\mathsf{dcoe}\ A\ B\ p)\ i)$$

Proof. The idea is that, as $\mathsf{duahom}\ A\ B\ (\mathsf{dcoe}\ A\ B\ p)\ i$ is a glue type with base B (i.e. $p\ \mathbb{1}_{\mathbb{2}}$), we can take any term in $p\ i$ and coerce it to a term in $p\ \mathbb{1}_{\mathbb{2}}$ to define our function. We construct the function as $\lambda i\ x.\mathsf{cov}^{i \mapsto \mathbb{1}_{\mathbb{2}}} p\ x$. Note that we used a different covariant filler that goes from i to $\mathbb{1}_{\mathbb{2}}$. This follows from our standard covariant filler that goes from $\mathbb{0}_{\mathbb{2}}$ to i by augmenting the path with a connection: $\lambda x.p(x \sqcup i)$. ⊠

4.2 Cobar Modal Types

Next, we will restrict $\mathsf{U}_{\mathsf{cov}}$ to a subuniverse for which we can show that $\mathsf{dua}\eta\mathsf{fun}$ is an equivalence (we do not have countermodel showing definitively that $\mathsf{dua}\eta\mathsf{fun}$ is not an equivalence for $\mathsf{U}_{\mathsf{cov}}$, but we have not been able to prove it). We do this using a modality which picks out those types that are equivalent to their *cobar construction*. The definitions and many of the theorems in this section follow Coquand, Sattler and Ruch [28]; we also check that this work applies to our setting and prove a few extra lemmas needed for our use of theses ideas. Coquand, Sattler and Ruch abstract many of the definitions and lemmas to the level of an arbitrary left-exact (lex) modality on types, and we have formalized this part of the construction in Agda. The parts that are cobar-specific are postulated as axioms.

We axiomatize the cobar operator as is shown in Figure 4.1. The main idea is that we add a new endofunctor D on fibrant types which comes equipped with a natural transformation $\eta : \mathrm{id}_{\mathsf{U}_{\mathsf{Kan}}} \to D$. Using this, we make a restricted version of our covariant universe containing those covariant types A for which $\eta_A : A \to \mathsf{D}A$ is an equivalence, and call these types cobar modal. We show that all of the basic type formers preserve the property of being cobar modal. With the added structure of this equivalence for every type, we are able to

$$
\begin{array}{rcl}
\mathsf{D} & : & \mathsf{U}_{\mathsf{Kan}} \to \mathsf{U}_{\mathsf{Kan}} \\
\eta & : & \{A : \mathsf{U}_{\mathsf{Kan}}\} \to [\![A]\!]_{\mathsf{Kan}} \to [\![\mathsf{D}A]\!]_{\mathsf{Kan}} \\
\mathsf{L} & : & [\![\mathsf{D}(\mathsf{U}_{\mathsf{Kan}})]\!]_{\mathsf{Kan}} \to \mathsf{U}_{\mathsf{Kan}} \\
\\
\tilde{\mathsf{D}} & : & \{A : \mathsf{U}_{\mathsf{Kan}}\} \to ([\![A]\!]_{\mathsf{Kan}} \to \mathsf{U}_{\mathsf{Kan}}) \\
 & & \to [\![\mathsf{D}A]\!]_{\mathsf{Kan}} \to \mathsf{U}_{\mathsf{Kan}} \\
\tilde{\mathsf{D}}\ B & := & \mathsf{L} \circ \mathsf{Dfun}(B) \\
\\
\mathsf{isModal} & : & \mathsf{U}_{\mathsf{Kan}} \to \mathsf{U}_{\mathsf{Kan}} \\
\mathsf{isModal}\ A & := & \mathsf{isEquiv}\ \eta_A \\
\\
\mathsf{Dfun} & : & \{A\ B : \mathsf{U}_{\mathsf{Kan}}\} \to ([\![A]\!]_{\mathsf{Kan}} \to [\![B]\!]_{\mathsf{Kan}}) \\
 & & \to [\![\mathsf{D}A]\!]_{\mathsf{Kan}} \to [\![\mathsf{D}B]\!]_{\mathsf{Kan}} \\
\mathsf{Dcomp} & : & \{A\ B\ C : \mathsf{U}_{\mathsf{Kan}}\} \to (g : [\![B]\!]_{\mathsf{Kan}} \to [\![C]\!]_{\mathsf{Kan}}) \\
 & & \to (f : [\![A]\!]_{\mathsf{Kan}} \to [\![B]\!]_{\mathsf{Kan}}) \\
 & & \to \mathsf{Dfun}(g \circ f) = \mathsf{Dfun}(g) \circ \mathsf{Dfun}(f) \\
\mathsf{Did} & : & (A : \mathsf{U}_{\mathsf{Kan}}) \\
 & & \to \mathsf{Dfun}(\lambda a : [\![A]\!]_{\mathsf{Kan}}.a) = \lambda a : [\![\mathsf{D}A]\!]_{\mathsf{Kan}}.a \\
\eta\mathsf{nat} & : & \{A\ B : \mathsf{U}_{\mathsf{Kan}}\} \to (f : [\![A]\!]_{\mathsf{Kan}} \to [\![B]\!]_{\mathsf{Kan}}) \\
 & & \to \mathsf{Dfun}(f) \circ \eta_A = \eta_B \circ f \\
\eta\mathsf{path} & : & (A : \mathsf{U}_{\mathsf{Kan}}) \\
 & & \to \mathsf{Path}_{[\![\mathsf{D}A]\!]_{\mathsf{Kan}} \to [\![\mathsf{D}^2A]\!]_{\mathsf{Kan}}}\ (\mathsf{Dfun}(\eta_A),\ \eta_{\mathsf{D}A}) \\
\mathsf{Leq} & : & \mathsf{L} \circ \eta_{\mathsf{U}_{\mathsf{Kan}}} = \mathsf{D} \\
\mathsf{Lmodal} & : & (A : [\![\mathsf{D}(\mathsf{U}_{\mathsf{Kan}})]\!]_{\mathsf{Kan}}) \to \mathsf{isModal}\ (\mathsf{L}A) \\
\\
\mathsf{D}\Sigma\mathsf{snd} & : & \{A : \mathsf{U}_{\mathsf{Kan}}\} \to \{B : [\![A]\!]_{\mathsf{Kan}} \to \mathsf{U}_{\mathsf{Kan}}\} \\
 & & \to \Pi p : [\![\mathsf{D}(\Sigma a : A.B\ a)]\!]_{\mathsf{Kan}}.[\![\tilde{\mathsf{D}}B\ (\mathsf{Dfun}\ \mathsf{fst}\ a)]\!]_{\mathsf{Kan}} \\
\mathsf{D}\Sigma\mathsf{eq} & : & \{A : \mathsf{U}_{\mathsf{Kan}}\} \to (B : [\![A]\!]_{\mathsf{Kan}} \to \mathsf{U}_{\mathsf{Kan}}) \\
 & & \to \mathsf{isIso}\ (\lambda x.(\mathsf{Dfun}\ \mathsf{fst}\ x, \mathsf{D}\Sigma\mathsf{snd}\ x))
\end{array}
$$

Figure 4.1: Axioms and definitions for descent operator

show that a function is an equivalence of cobar modal types if at every object in $\square_{\mathrm{Ded}}$ it is an equivalence of cubical sets, from which we can complete the construction of directed univalence for the universe of covariant cobar modal types.

4.2.1 Universes and Closure Properties of Lex Modal Types

First, we formally define what it is to be a lex modality.

Definition 4.2.1. A lex operator on the universe $\mathsf{U}_{\mathsf{Kan}}$ is an endofunctor $\mathsf{D} : \mathsf{U}_{\mathsf{Kan}} \to \mathsf{U}_{\mathsf{Kan}}$ accompanied by the structure axiomatized in Figure 4.3 omitting the axiom $\eta\mathsf{path}$. A lex

modality, also called a descent data operation, is lex operator that also satisfies the axiom $\eta\mathsf{path}$, and for which $\mathsf{Dfun}(\eta_A)$ is an equivalence for all types A in $\mathsf{U}_{\mathsf{Kan}}$.

We now show that the modal types defined by a generic lex modality are closed under the following:

Lemma 4.2.2 (`descent.Lex.QisStack-to-isStack`, [28, Lemma 10])**.** *Given a Kan type A, A is modal if there is a patch function $p_A : \mathsf{D}A \to A$ such $p_A \circ \eta_A : A \to A$ is path equal to the identity function on A.*

Proof. To complete the equivalence, all we need to show is that $\eta_A \circ p_A : \mathsf{D}A \to \mathsf{D}A$ is path equal to the identity. Let $e : \mathsf{Path}_{A\to A}\,(p_A \circ \eta_A, \mathrm{id}_A)$ be the name of the path given as a hypothesis. First, applying functorality of D to e gives us a path from $\mathsf{Dfun}\ p_A \circ \mathsf{Dfun}\ \eta_A$ to $\mathrm{id}_{\mathsf{D}A}$. Using $\eta\mathsf{path}\ A : \mathsf{Path}_{[\![\mathsf{D}A]\!]_{\mathsf{Kan}}\to[\![\mathsf{D}^2A]\!]_{\mathsf{Kan}}}\,(\mathsf{Dfun}(\eta_A),\ \eta_{\mathsf{D}A})$, we can also define a path from $\mathsf{Dfun}\ p_A \circ \mathsf{Dfun}\ \eta_A$ to $\mathsf{Dfun}\ p_A \circ \eta_{\mathsf{D}A}$. As η is natural, $\mathsf{Dfun}\ p_A \circ \eta_{\mathsf{D}A}$ equals $\eta_A \circ p_A$. As such, composing the inverse of the second path we defined with the first gives us a path connecting $\eta_A \circ p_A$ to $\mathrm{id}_{\mathsf{D}A}$, concluding the equivalence and therefore that A is modal. ⊠

Lemma 4.2.3 (`descent.Pi.ΠisStack`, [28])**.** *Given a Kan type A and a Kan fibration $B : [\![A]\!]_{\mathsf{Kan}} \to \mathsf{U}_{\mathsf{Kan}}$, if the fibers of B are modal then $\Pi x : A.B\ x$ is modal.*

Proof. We begin by defining the patch function $p_\Pi : \mathsf{D}(\Pi x : A.B\ x) \to \Pi x : A.B\ x$. Let $p_B : \Pi x : A.\mathsf{D}(B\ x) \to B\ a$ be the family of patch functions we have for B. As D is functorial, given an $a : A$ we can apply it to the application function $\lambda f.f\ a : (\Pi x : A.B\ x) \to B\ a$ to give us a function from $\mathsf{D}(\Pi x : A.B\ x) \to \mathsf{D}(B\ a)$. We can then compose with $p_B\ a$ to land in $B\ a$, giving us the function we ultimately desire.

$$p_\Pi\ f\ a := (p_B\ a \circ \mathsf{Dfun}(\lambda f'.f'\ a))\ f$$

By Lemma 4.2.2, the only thing left to show is that there is a path between the identity function and $p_\Pi \circ \eta_{\Pi x:A.B\ x}$. First, using function extensionality and naturality of η along

the function $\lambda f'.f'\ a : \Pi x : A.B\ x \to B\ a$, we know that $p_\Pi \circ \eta_{\Pi x:A.B\ x}$ is strictly equal to $\lambda a.p_B\ a\ (\eta_{B\ a}\ (f\ a))$. As the fibers of B are all modal, we have a path from $p_B\ a \circ \eta_{B\ a}$ to the identity function on $B\ a$ for every a, and thus by function extensionality for paths in Π-types this lifts to give us the desired path from $p_\Pi \circ \eta_{\Pi x:A.B\ x}$ to the identity function in the Π-type.

$\boxtimes$

Lemma 4.2.4 (`descent.Lex.DΣ-Path`, [28])**.** *Given a Kan type A and a Kan fibration $B : A \to \mathsf{U}_{\mathsf{Kan}}$, if the fibers of B are modal, there is a path between the types $\Sigma a : A.B\ a$ and $\Sigma a : A.\tilde{\mathsf{D}}B\ (\eta\ a)$ in $\mathsf{U}_{\mathsf{Kan}}$.*

Proof. First, let us expand the definition of $\tilde{\mathsf{D}}$ in $\Sigma a : A.\tilde{\mathsf{D}}B\ (\eta\ a)$, which gives us the type $\Sigma a : A.(\mathsf{L} \circ \mathsf{Dfun}(B) \circ \eta_A)\ a$. By naturality of η, the function $\mathsf{L} \circ \mathsf{Dfun}(B) \circ \eta_A$ equals $\mathsf{L} \circ \eta_{\mathsf{U}_{\mathsf{Kan}}} \circ B$, and as the axiom Leq states $\mathsf{L} \circ \eta_{\mathsf{U}_{\mathsf{Kan}}} = \mathsf{D}$, the Σ-type we began with is equal to $\Sigma a : A.(\mathsf{D} \circ B)\ a$. As the fibers of B are modal, we have a path between $\mathsf{D}\ (B\ a)$ for every a, and thus we can use these with function extensionality of paths to get a path between the fibrations B and $D \circ B$, and then apply the function $\lambda X.\Sigma a : A.X\ a$ to said path to conclude our lemma. $\boxtimes$

Lemma 4.2.5 (`descent.Sigma.ΣisStack`, [28, Proposition 7])**.** *If a Kan type A is modal and the fibers of a Kan fibration $B : A \to \mathsf{U}_{\mathsf{Kan}}$ are modal, then $\Sigma a : A.B\ a$ is modal.*

Proof. From Lemma 4.2.4, we know there is a path between $\Sigma a : A.B\ a$ and $\Sigma a : A.\tilde{\mathsf{D}}B\ (\eta\ a)$ and thus by univalence an equivalence between them. As A is modal η_A is an equivalence, and thus $\Sigma a : A.\tilde{\mathsf{D}}B\ (\eta\ a)$ is equivalent to $\Sigma a : \mathsf{D}A.\tilde{\mathsf{D}}B\ a$. Finally, the axiom $\mathsf{DΣeq}$ states that this is isomorphic to $\mathsf{D}(\Sigma a : A.B\ a)$, and therefore $\Sigma a : A.B\ a$ is equivalent to $\mathsf{D}(\Sigma a : A.B\ a)$. Lastly, we can see the above equivalence is indeed along $\eta_{\Sigma a:A.B\ a}$ by naturality of η in conjunction with $\mathsf{DΣeq}$, and thus the Σ-type is modal. $\boxtimes$

We now can describe the modal versions of our universes.

Definition 4.2.6. We define the following universes:

- The universe of cobar modal types $\mathsf{U}_{\mathsf{cobar}}$ is defined as $\Sigma X : \mathsf{U}_{\mathsf{Kan}}.\mathsf{isModal}\ X$;
- The universe of covariant cobar modal types $\mathsf{U}_{\mathsf{covCobar}}$ is defined as $\Sigma X : \mathsf{U}_{\mathsf{cov}}.\mathsf{isModal}\ X$;
- The universe of inner cobar modal types $\mathsf{U}_{\mathsf{innerCobar}}$ is defined as $\Sigma X : \mathsf{U}_{\mathsf{inner}}.\mathsf{isModal}\ X$.

Lemma 4.2.7 (`descent.Stack.UCobar-isStack`, [28, Proposition 11])**.** *The universe of cobar modal types* $\mathsf{U}_{\mathsf{cobar}}$ *is modal.*

Proof. First, note that from the axiom Lmodal we know that the fibers of L are modal. As such, we can define a function $\alpha : [\![\mathsf{D}\ \mathsf{U}_{\mathsf{Kan}}]\!]_{\mathsf{Kan}} \to \mathsf{U}_{\mathsf{cobar}}$ such that the first component is given by L. Using this, we then define our candidate patch function $p : \mathsf{D}\mathsf{U}_{\mathsf{cobar}} \to \mathsf{U}_{\mathsf{cobar}}$ as the composition $\alpha \circ \mathsf{D}\pi_1$.

To conclude $\mathsf{U}_{\mathsf{cobar}}$ is modal, we now must show that $p \circ \eta_{\mathsf{U}_{\mathsf{cobar}}}$ is path equal to the identity function on $\mathsf{U}_{\mathsf{cobar}}$. As being modal is a homotopy proposition, it suffices to ignore the proof witness and just show that $\pi_1 \circ \eta_{\mathsf{U}_{\mathsf{cobar}}}$ is path equal to the $\pi_1 : \mathsf{U}_{\mathsf{cobar}} \to \mathsf{U}_{\mathsf{Kan}}$. First, we can unfold our definitions and use equations from our axioms to rewrite the function.

$$\begin{aligned} \pi_1 \circ p \circ \eta_{\mathsf{U}_{\mathsf{cobar}}} &= \pi_1 \circ \alpha \circ \mathsf{D}\pi_1 \circ \eta_{\mathsf{U}_{\mathsf{cobar}}} \\ &= \mathsf{L} \circ \mathsf{D}\pi_1 \circ \eta_{\mathsf{U}_{\mathsf{cobar}}} \\ &= \mathsf{L} \circ \eta_{\mathsf{U}_{\mathsf{Kan}}} \circ \pi_1 \\ &= \mathsf{D} \circ \pi_1 \end{aligned}$$

As every type A in $\mathsf{U}_{\mathsf{cobar}}$ is cobar modal, univalence allows us to transform these equivalences into paths, and in doing so define a path between $\pi_1 : \mathsf{U}_{\mathsf{cobar}} \to \mathsf{U}_{\mathsf{Kan}}$ and $\mathsf{D} \circ \pi_1$, concluding our proof. ⊠

4.2.2 The Cobar Construction

Next, we work externally (so this part has not been formalized) and check that axioms in Figure 4.1 are true. Coquand, Sattler and Ruch define the cobar operator by first defining a type operator E [28].

Definition 4.2.8. Given a Kan type A in a context Γ, the type $\mathsf{E}A$ in context Γ has terms as follows: For every X in $\square_{\mathrm{Ded}}$ and ρ in $\Gamma(X)$, and element u in $\mathsf{E}A\rho(X)$ is given by a family of elements $u(f)$ of $A(\rho f)(Y)$ indexed by morphisms $f \in \square_{\mathrm{Ded}}(Y, X)$. It is equipped with a natural transformation $\alpha_A : A \to \mathsf{E}A$ (fibered over each element ρ in Γ) sending a in $A\rho(X)$ to the family $f \mapsto af$.

In particular, notice that for X in $\square_{\mathrm{Ded}}$, an element of $\mathsf{E}^n A(X)$ is a function that takes as input an n-chain of composable morphisms with codomain X and returns an element of $A(Y)$ where Y is the domain of the chain.

In the definition of the cobar operator, we will need the following: Given a list $(i_0, i_1, \ldots, i_n) : \mathbb{I}^{n+1}$, the formula $\delta_{n+1}(i_0, i_1, \ldots, i_n)$ is the $n+1$-ary disjunction

$$(i_0 = \mathbb{1}_{\mathbb{I}}) \vee (i_1 = \mathbb{1}_{\mathbb{I}}) \vee \ldots \vee (i_{n-1} = \mathbb{1}_{\mathbb{I}}) \vee (i_n = \mathbb{1}_{\mathbb{I}})$$

We then define a family of morphisms $s_k : \mathbb{I}^{n+1} \to \mathbb{I}^n$ for $1 \leq k \leq n+1$ where $s_k(\vec{i})$ omits i_k.

Definition 4.2.9. Given a Kan type A in a context Γ, the type $\mathsf{D}A$ in context Γ has terms as follows: For every X in $\square_{\mathrm{Ded}}$ and ρ in $\Gamma(X)$, and element u in $\mathsf{D}A\rho(X)$ is given by a family of elements $u(i_0, \ldots, i_n)$ of $\mathsf{E}^{n+1}A$ defined over $\delta_n(i_0, \ldots, i_n)$ indexed by lists of interval variables $(i_0, \ldots, i_n) : \mathbb{I}^{n+1}$ for every natural number n. These families must also satisfy the following condition that $u(\vec{i}) = E^k(\alpha)u(s_k\ \vec{i})$ on $i_k = \mathbb{0}_{\mathbb{I}}$.

We define the natural transformation $\eta_A : A \to \mathsf{D}A$ such that $(\eta_A\ a)(i_0, \ldots, i_n) := \alpha^{n+1}a$.

The contents of the family $u : \mathsf{D}A(X)$ conceptually includes a choice of a term in $A(Y)$ for every element of $A(X)$ and every morphism f from Y to X with the idea that the selected term is to approximate the morphism action on the term in $A(X)$. This then is accompanied with path structure indicating the choice is coherent with both the actual morphism action of the presheaf and the way the choice for the action of f interacts with the other choices made by u via composition. This works out in such a way that natural transformations into

$\mathsf{D}A$ correspond precisely with the transformation into A that are homotopy coherent with respect to $\square_{\text{Ded}}$.[1]

While the details of the definition are quite technical, the summarized idea is that for every Dedekind cube X a term in $\mathsf{D}A(X)$ is a family u_n indexed by $n : \mathbb{N}$, and each u_n is a function that takes as input an $n+1$-length chain of morphisms in $\square_{\text{Ded}}$ with codomain X, and returns a partial $n+1$-cube with shape given by δ_n in $A(Y)$ where Y is the domain of the chain. These families are constrained so that the boundaries of the partial $n+1$-cubes given by u_n are built out of u_k for $k \leq n+1$ (with some of the interval variables evaluated to $\mathbb{1}_\mathbb{I}$, ensuring they are of smaller dimension) where the u_k are evaluated at all of the k-chains given by all possibilities of replacing adjacent morphisms of $n+1$-chain with their compositions, or by applying the morphism action of morphisms at the domain of the chain to u_k evaluated at (compositions of) the remainder of the chain.

Let us unfold the definition and work through what a term $u : \mathsf{D}A(X)$ precisely contains for the first few dimensions. First, for $n = 0$, we have a family indexed by morphisms $f : \square_{\text{Ded}}(Y, X)$, and an interval variable i_0. As the interval variables are restricted by δ, for this case we know that $i_0 = \mathbb{1}_\mathbb{I}$, and thus despite the presence of the variable we actually just have a single point: $u(\mathbb{1}_\mathbb{I})(f)$ in the presheaf $A(Y)$.

Moving up to $n = 1$ in the family, u is now indexed by two interval variables, i_0 and i_1, and two morphisms $f : \square_{\text{Ded}}(Y, X)$ and $g : \square_{\text{Ded}}(Z, Y)$. The partial square given by $\lambda i_0\ i_1.u(i_0, i_1)(f, g)$ in the cubical set $A(Z)$ is only defined along the sides $i_0 = \mathbb{1}_\mathbb{I} \vee i_1 = \mathbb{1}_\mathbb{I}$. Noting that here $\alpha\ u(\mathbb{1}_\mathbb{I})(f) = u(\mathbb{1}_\mathbb{I})(f)g$ and that $E(\alpha)u(\mathbb{1}_\mathbb{I})(g) = u_1(\mathbb{1}_\mathbb{I})(f \circ g)$, the boundary constrains indicate it consists of the following picture in $A(Z)$.

$$\begin{array}{ccc} u(\mathbb{1}_\mathbb{I})(f)g & \xrightarrow{u(i_0,\mathbb{1}_\mathbb{I})(f,g)} & u(\mathbb{1}_\mathbb{I},\mathbb{1}_\mathbb{I})(f,g) \\ & & \Big| {\scriptstyle u(\mathbb{1}_\mathbb{I},i_1)(f,g)} \\ & & u(\mathbb{1}_\mathbb{I})(f \circ g) \end{array}$$

[1] Homotopy coherent transformations are a weakened notion of natural transformations in which the naturality squares only must be satisfied up to homotopy.

In particular, we have the point $u(\mathbb{1}_{\mathbb{I}})(f)$ in $A(Y)$ along with two new points $u(\mathbb{1}_{\mathbb{I}}, \mathbb{1}_{\mathbb{I}})(f, g)$ and $u(\mathbb{1}_{\mathbb{I}})(f \circ g)$ in $A(Z)$, along with a path $u(i_0, \mathbb{1}_{\mathbb{I}})(f, g)$ connecting the action of the morphism g on $u(\mathbb{1}_{\mathbb{I}})(f)$ to $u(\mathbb{1}_{\mathbb{I}}, \mathbb{1}_{\mathbb{I}})(f, g)$ and a path $u(\mathbb{1}_{\mathbb{I}}, i_1)(f, g)$ connecting $u(1)(f \circ g)$ to $u(\mathbb{1}_{\mathbb{I}}, \mathbb{1}_{\mathbb{I}})(f, g)$.

Lastly, at $n = 2$, we will consider u evaluated over the three morphisms $f : \square_{\text{Ded}}(Y, X)$, $g : \square_{\text{Ded}}(Z, Y)$, and $h : \square_{\text{Ded}}(W, Z)$. First, as has been the case so far and is always the case, $u(\mathbb{1}_{\mathbb{I}}, \mathbb{1}_{\mathbb{I}}, \mathbb{1}_{\mathbb{I}})(f, g, h)$ picks out a new point in the domain $A(W)$ for the 3-chain (f, g, h). The rest of the structure in the partial cube is then connecting this point to the lower-dimensional components of u for the various chains given by composing adjacent elements in the chain (e.g. $u(\mathbb{1}_{\mathbb{I}}, \mathbb{1}_{\mathbb{I}})(f, g \circ h)$ and $u(\mathbb{1}_{\mathbb{I}}, i_2)(f, g \circ h)$) along with the actual action of the morphisms moving the lower-dimensional parts of u built from f and g into $A(W)$ (e.g. $u(\mathbb{1}_{\mathbb{I}})(f)(g \circ h)$ and $u(\mathbb{1}_{\mathbb{I}}, i_2)(f, g)h$). More specifically, $u(i_0, i_1, i_2)(f, g, h)$ consists of three conjoined squares: one for each disjunct in $i_0 = \mathbb{1}_{\mathbb{I}} \vee i_1 = \mathbb{1}_{\mathbb{I}} \vee i_2 = \mathbb{1}_{\mathbb{I}}$. We draw each square individually in Figure 4.2.

While just glancing at all three squares initially looks a lot like random chaos, there is a pattern within the madness. Moving from the bottom left to to top right of each square represents the six distinct ways using only f, g and h to modify a 1-chain with codomain X into the 3-chain (f, g, h), with the chains being those supplied as arguments to u at each vertex. In particular, there are three 1-chains with codomain X possible, one corresponding to each square: f, $f \circ g$, and $f \circ g \circ h$. We can then increase the size of the chains by either adding a new morphism or by splitting a composition: as we see in the second square, the path along the left of the square modifies the 1-chain $f \circ g$ to the 2-chain (f, g) by splitting the composition, while the path along the bottom of the square adds to the chain by introducing h, turning the 1-chain $f \circ g$ into the 2-chain $(f \circ g, h)$.

In general, $u_n(\vec{i})(\vec{f})$ consists of the $n + 1$-many n-cubes where every n-cube's $\vec{\mathbb{0}}_{\mathbb{I}}$-vertex corresponds to one of the 1-chains taken by composing morphisms from the beginning of $\vec{f}$, and every cube's $\vec{\mathbb{1}}_{\mathbb{I}}$-vertex corresponds to the entire n-chain $\vec{f}$. The paths along the boundary

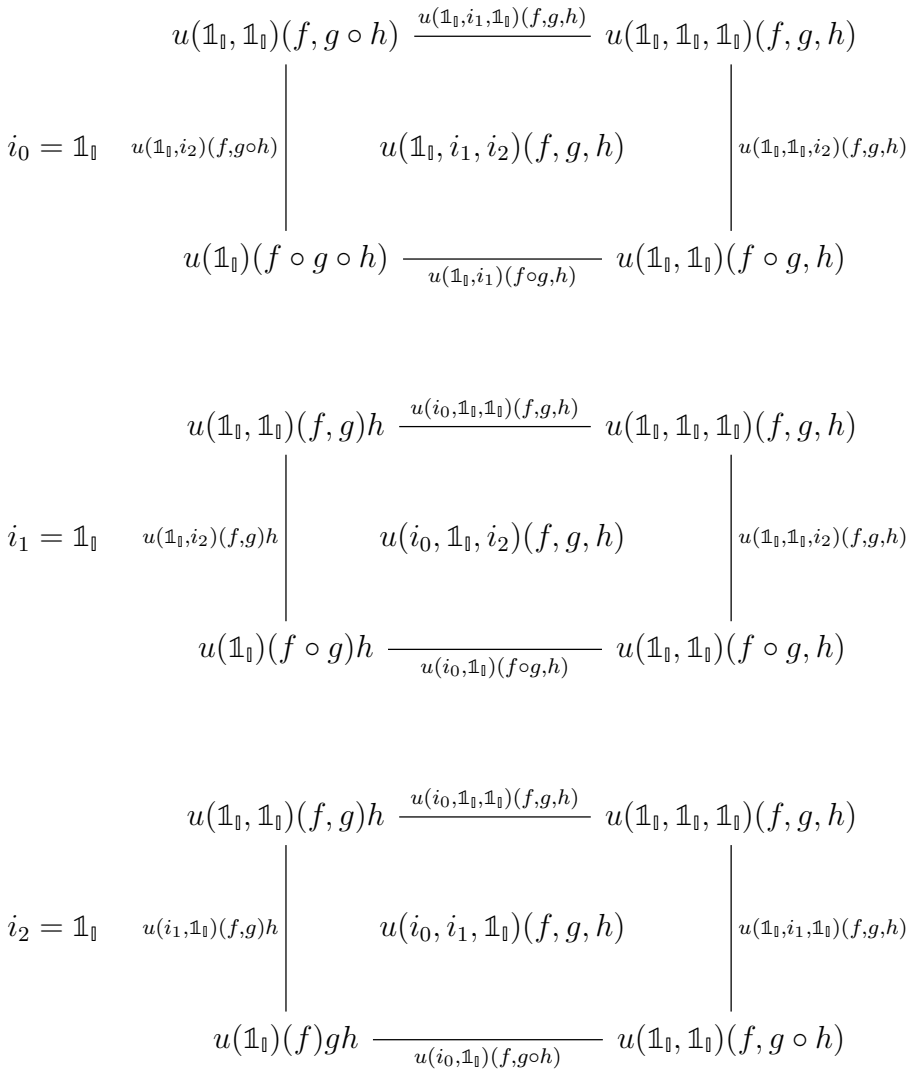


Figure 4.2: Depiction of $u(i_0, i_1, i_2)(f, g, h)$

of each cube relate to the sequences of the various moves one can make that increase the length of the chain by one (by either splitting a composition or adding a [composition of] morphism[s] to the end of the chain) to ultimately transform the 1-chains into the n-chain $\vec{f}$.

Proposition 4.2.10. *For any Kan type A, there is a path in $\mathsf{D}A \to \mathsf{D}^2A$ between $\eta_{\mathsf{D}A}$ and* $\mathsf{Dfun}\ \eta_A$.

Proof. As is described in [28, Proposition 20], an element of D^2A is of the form $v_{m,n}(i_1, \ldots, i_m)(j_1, \ldots, j_n)$ in $E^{m+n+2}A$ satisfying a number of equations. In the pres-

ence of connections, a homotopy from $\eta_{\mathsf{D}A}(u)_{m,n}$ to $(\mathsf{Dfun}\ \eta_A)(u)_{m,n}$ is given by the composition of $\lambda k : \mathbb{I}.v^k_{m,n}$ and $\lambda k : \mathbb{I}.w^k_{m,n}$ where $v^k_{m,n}(\vec{i})(\vec{j}) := u_{m+n+1}(\vec{i} \wedge k, \vec{j})$ and $w^k_{m,n}(\vec{i})(\vec{j}) := u_{m+n+1}(\vec{i}, k \wedge \vec{j})$.

Given our work is agnostic to the choice of cube category $\square_{\text{path}}$ for paths, we would like a proof that does not depend on the presence of connections; however, unlike in [28], we restrict our functor D to only be over fibrant types, and thus we can use path induction to replace the use of connections in their proof. Given some term $a : \mathbb{I} \to A$ for a Kan type A, we can construct a term $a_\wedge : \Pi(i, j) : \mathbb{I}^2.A[i = \mathbb{0}_\mathbb{I} \mapsto a\ \mathbb{0}_\mathbb{I}, i = \mathbb{1}_\mathbb{I} \mapsto a\ j, j = \mathbb{0}_\mathbb{I} \mapsto a\ \mathbb{0}_\mathbb{I}, j = \mathbb{1}_\mathbb{I} \mapsto a\ i, i = j \mapsto a\ i]$ by path induction on a, as the extension type is fibrant. Having defined these, we build a homotopies with the same boundaries as $v^k_{m,n}$ and $w^k_{m,n}$ by induction on m and n, iteratively using this constructions in place of the connection.

To construct $v^k_{m,n}$ where the length of $\vec{i}$ is m and that of $\vec{j}$ is n, we first do induction over an auxiliary number $z \leq m$ to define a term v_z of type $\mathbb{I}^{z+m+n+1} \to \mathsf{E}^{m+n+2}A$. when $z = 0$, we simply define the term to be $v_0 := \lambda\vec{x} : \mathbb{I}^{m+n+1}.u_{m+n+1}(\vec{x})$. Now, assume we have defined $v_z : \mathbb{I}^{z+m+n+1} \to \mathsf{E}^{m+n+2}A$. To define the term $v_{z+1} : \mathbb{I}^{z+1+m+n+1} \to \mathsf{E}^{m+n+2}A$, we isolate and curry the variable x_{z+1} from $\vec{x}$ (where x_{z+1} will ultimately correspond to the $(z+1)$-th element in the m-length list $\vec{i}$), and then use the $\wedge$-connection construction on $\lambda.x_{z+1}.\lambda\vec{x} \setminus x_{z+1}.w_z(\vec{x})$, adding a new dimension variable for the connection at position x_{z+1} and resulting in v_{z+1} of the desired type. Given we can do this construction for any $z \leq m$, we in particular can do it for m itself, resulting in a term $v_m : \mathbb{I}^{2m+n+1} \to E^{m+n+2}A$. Lastly, we repeat the above construction using n instead of m and working from the variables indexed at the end of $\vec{x}$ instead of the beginning to define a term $w_n : \mathbb{I}^{m+2n+1} \to E^{m+n+2}A$.

To finish, we abstract v_m and w_n over a fresh dimension variable k and substitute back in $\vec{i}$, $\vec{j}$ and k in the correct positions of its arguments corresponding to the definitions of $v^k_{m,n}$ and $w^k_{m,n}$ that use connections, resulting in two cubes that have identical boundaries. Given this construction results in terms with identical boundaries as those built in [28, Proposition 20] using connections, the families defined using this also satisfies the constraints required to be

an element of $\mathsf{D}^2 A$, and thus the two are well typed and can be composed to define the path we set out to build. ⊠

Theorem 4.2.11 (Corollary 19 in [28])**.** *The cobar operator is a lex operator.*

Proof. The proof that cobar operator is a lex operator (and thus satisfies the axioms in Figure 4.3 excluding $\eta\mathsf{path}$) can be found in [28]. The only proof from [28] that depends on having connections in the undirected cube category is that corresponding to Proposition 4.2.10, which we have proven without assuming the presence of connections above. ⊠

Lemma 4.2.12 (Corollary 24 in [28])**.** *The cobar operator is a lex modality as defined in [59] and [65].*

Proof. As with the proof of Theorem 4.2.11, the proof in [28] works in our setting modulo the assumption of connections in the undirected cube category being used in their proof corresponding to Proposition 4.2.10. That being said, given we have provided an alternative proof, their proof of cobar being a lex modality applies here as well. Do note that this implies the cobar operator additionally satisfies the $\eta\mathsf{path}$ axiom, and thus satisfies all of the axioms listed in Figure 4.3. ⊠

To complete the proofs that cobar modal types are closed under path and morphism types, we add a few isomorphisms and equations for η to our model, shown in Figure 4.3.

Lemma 4.2.13. *For any representable $\Psi = \mathbb{I}^n \times \mathbb{2}^m$ and Kan fibration $A : \Psi \to \mathsf{U}_{\mathsf{Kan}}$ in context Γ, $\mathsf{D}(\Pi x : \Psi.A\ x)$ is isomorphic to $\Pi x : \Psi.\mathsf{D}(A\ x)$.*

Proof. As our representables are all give by products of $\mathbb{I}$ and $\mathbb{2}$, it is sufficient to just show it holds for each interval. First, consider $\mathsf{E}(\Pi i : \mathbb{I}.A\ i)$. For each $X \in \square_{\mathrm{Ded}}$, $\rho \in \Gamma(X)$, $u \in \mathsf{E}(\Pi i : \mathbb{I}.A\ i)\rho$ and $f \in \square_{\mathrm{Ded}}(Y, X)$, $u(f)$ is a term in $\Pi i : \mathbb{I}.(A\ i)\rho f$. We can reorder the arguments to get an element of $\Pi i : \mathbb{I}.\mathsf{E}(A\ i)$ by $(i, X, f) \mapsto u(X, f, i)$, which is clearly an isomorphism. Similarly, for $\mathsf{E}(\Pi i : \mathbb{2}.A\ i)$, for each $X \in \square_{\mathrm{Ded}}$, $u \in \mathsf{E}(\Pi i : \mathbb{2}.A\ i)\rho$ and $f \in \square_{\mathrm{Ded}}(Y, X)$, $u(f)$ is a term in $\Pi i : \mathbb{2}.(A\ i)\rho f$. This is equivalent to considering $u(f \times \mathrm{id}_{\mathbb{2}})$

$$
\begin{array}{rcl}
\mathsf{Ddua} & : & \forall\{A\ B : \mathsf{U}_{\mathsf{Kan}}\}\{f : [\![A]\!]_{\mathsf{Kan}} \to [\![B]\!]_{\mathsf{Kan}}\}\{i : \mathbb{2}\} \\
& & \to \mathsf{D}(\mathsf{duahom}'\ A\ B\ f\ i) \\
& & \to \mathsf{duahom}'\ (\mathsf{D}A)\ (\mathsf{D}B)\ (\mathsf{Dfun}\ f)\ i \\
\mathsf{Ddua}\ g & := & \mathsf{glue}[i = \mathbb{0}_2 \mapsto g, i = \mathbb{1}_2 \mapsto \mathsf{Dfun}\ \mathsf{unglue}\ g] \\
& & \quad (\mathsf{Dfun}\ \mathsf{unglue}\ g)
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{Ddua\text{-}iso} & : & \mathsf{isIso}\ \_\ \_\ \mathsf{Ddua} \\
\mathsf{DPath\text{-}iso} & : & \{A : \mathsf{U}_{\mathsf{Kan}}\} \to (a_0\ a_1 : [\![A]\!]_{\mathsf{Kan}}) \\
& & \to \mathsf{Iso}\ (\mathsf{DPath}_{\mathsf{D}A}\ \eta a_0\ \eta a_1)\ (\mathsf{D}(\mathsf{DPath}_A\ a_0\ a_1)) \\
\mathsf{DHom\text{-}iso} & : & \{A : \mathsf{U}_{\mathsf{Kan}}\} \to (a_0\ a_1 : [\![A]\!]_{\mathsf{Kan}}) \\
& & \to \mathsf{Iso}\ (\mathsf{DHom}_{\mathsf{D}A}\ \eta a_0\ \eta a_1)\ (\mathsf{D}(\mathsf{DHom}_A\ a_0\ a_1)) \\
\mathsf{Path}\text{-}\eta\text{-}\mathsf{eq} & : & \forall\{A : \mathsf{U}_{\mathsf{Kan}}\}\{a_0\ a_1 : [\![A]\!]_{\mathsf{Kan}}\}(p : \mathsf{DPath}_A\ a_0\ a_1) \\
& & \to (\mathsf{DPath\text{-}iso}\ a_0\ a_1) \circ \eta_A \circ p = \eta_{\mathsf{DPath}_A\ a_0\ a_1}\ p \\
\mathsf{Hom}\text{-}\eta\text{-}\mathsf{eq} & : & \forall\{A : \mathsf{U}_{\mathsf{Kan}}\}\{a_0\ a_1 : [\![A]\!]_{\mathsf{Kan}}\}(p : \mathsf{DHom}_A\ a_0\ a_1) \\
& & \to (\mathsf{DHom\text{-}iso}\ a_0\ a_1) \circ \eta_A \circ p = \eta_{\mathsf{DHom}_A\ a_0\ a_1}\ p
\end{array}
$$

Figure 4.3: Additional axioms and definitions for cobar operator

a term in $A(Y \times \mathbb{2})$, and thus we can also pull out the $\mathbb{2}$ to the outside with Yoneda and rearrange the arguments to get a term in $\Pi i : \mathbb{2}.\mathsf{E}(A\ i)$. Given how $\mathsf{D}A$ is constructed using $\mathsf{E}A$, we can iterate this isomorphism and commute arguments with the indexing lists of interval variables to get the isomorphisms between $\mathsf{D}(\Pi i : \mathbb{I}.A\ i)$ and $\Pi i : \mathbb{I}.\mathsf{D}(A\ i)$, along with between $\mathsf{D}(\Pi i : \mathbb{2}.A\ i)$ and $\Pi i : \mathbb{2}.\mathsf{D}(A\ i)$. As this is simply reordering the arguments to the families, the equations required by D still hold. ⊠

Corollary 4.2.14. *For any representable $\Psi = \mathbb{I}^n \times \mathbb{2}^m$, Kan fibration $A : \Psi \to \mathsf{U}_{\mathsf{Kan}}$ in context Γ, cofibration β only depending on Ψ, and partial term $a : \Pi x : \Psi.\beta \to A\ x$, the type $\mathsf{D}(\Pi x : \Psi.(A\ x)[\beta \mapsto a\ x])$ is isomorphic to $\Pi x : \Psi.(\mathsf{D}(A\ x))[\beta \mapsto \eta_{A\ x}\ a\ x]$.*

Proof. Given how $\mathsf{D}(A\ x)$ is defined using E, we can unfold the definition and use the above isomorphism to see that $\mathsf{D}(\Pi x : \Psi.(A\ x)[\beta \mapsto a\ x])$ is a restriction of the type $\Pi x : \Psi.\mathsf{D}(A\ x)$ given by $\Pi x : \Psi.(\mathsf{D}(A\ x))[\beta \mapsto \eta_{A\ x}\ a\ x]$. ⊠

Corollary 4.2.15. *The axioms DPath-iso, DHom-iso, Path-η-eq and Hom-η-eq from Figure 4.3 hold.*

4.2.3 Universes and Closure Properties of Cobar Modal Types

Now we show that the modal types defined specifically using cobar are closed under the following:

Lemma 4.2.16 (`descent.Path.Path0-isStack`)**.** *If the fibers of a Kan fibration* $A : \mathbb{I} \to \mathsf{U}_{\mathsf{Kan}}$ *are cobar modal types, then for any terms* a_0 *in* $A\ \mathbb{0}_{\mathbb{I}}$ *and* a_1 *in* $A\ \mathbb{1}_{\mathbb{I}}$ *the type* $\mathsf{DPath}_A\, a_0\ a_1$ *is cobar modal.*

Proof. To construct the inverse to η, we use the isomorphism from Corollary 4.2.15 and construct an inverse to the function $\eta_A \circ - : \mathsf{DPath}_A\, a_0\ a_1 \to \mathsf{DPath}_{\mathsf{D}\circ A}\, (\eta_{A\ \mathbb{0}_{\mathbb{I}}}\ a_0)\ (\eta_{A\ \mathbb{1}_{\mathbb{I}}}\ a_1)$. Given a path p in $\mathsf{DPath}_{\mathsf{D}\circ A}\, (\eta_{A\ \mathbb{0}_{\mathbb{I}}}\ a_0)\ (\eta_{A\ \mathbb{1}_{\mathbb{I}}}\ a_1)$ and i in $\mathbb{2}$, the inverse just applies ${\eta_A^{-1}}_i$ to $p\ i$, using the fact that $A\ i$ is cobar modal to compose with the path in $\mathsf{Path}_{A\ i}\, ({\eta_A^{-1}}_i(\eta\ (p\ i)))\ (p\ i)$ when i equals $\mathbb{0}_{\mathbb{I}}$ or $\mathbb{1}_{\mathbb{I}}$. As $(\eta_{A\ i}, {\eta_A^{-1}}_i)$ form an adjoint equivalence for every i, it follows that this is also an adjoint equivalence and thus an equivalence. ⊠

Analogously, we have

Lemma 4.2.17 (`descent.Hom.Hom0-isStack`)**.** *If the fibers of a Kan fibration* $A : \mathbb{2} \to \mathsf{U}_{\mathsf{Kan}}$ *are cobar modal types, then for any terms* a_0 *in* $A\ \mathbb{0}_{\mathbb{2}}$ *and* a_1 *in* $A\ \mathbb{1}_{\mathbb{2}}$ *the type* $\mathsf{DHom}_A\, a_0\ a_1$ *is cobar modal.*

Proof. The proof that $\mathsf{DHom}_A\, a_0\ a_1$ is cobar modal is analogous to that of paths. ⊠

As with path and hom types, we add an isomorphism as an axiom in our model to facilitate the proof that cobar modal types are closed under duahom'.

Lemma 4.2.18. *Consider Kan types* A *and* B*, function* $f : [\![A]\!]_{\mathsf{Kan}} \to [\![B]\!]_{\mathsf{Kan}}$ *and* $i : \mathbb{2}$*. The function* Ddua *defined in Figure 4.3 is an isomorphism.*

Proof. Observe that, for every object $\Psi = \mathbb{I}^m \times \mathbb{2}^n$, the type $\Psi \to \mathsf{duahom}'\ A\ B\ f\ i$ is isomorphic to $\mathsf{duahom}'\ (\Psi \to A)\ (\Psi \to B)\ (f \circ -)\ i$. Unfolding the definition of D, we see the action of Ddua is an application of this isomorphism, and thus is an isomorphism itself. ⊠

Lemma 4.2.19 (`descent.FunGlue.Glue-isStack`)**.** *Given Kan types A and B, a function $f : [\![A]\!]_{\mathsf{Kan}} \to [\![B]\!]_{\mathsf{Kan}}$ and a term $x : \mathbb{2}$, if A and B are cobar modal types, then so is the type* $\mathsf{duahom}'\ A\ B\ f\ i$.

Proof. First, note that, as η is natural, the isomorphism in Lemma 4.2.18 allows us to conclude that $\eta_{\mathsf{duahom}'\ A\ B\ f\ i}$ factors through $\mathsf{duahom}'\ (\mathsf{D}A)\ (\mathsf{D}B)\ (\mathsf{Dfun}\ f)\ i$ via the function sending $\mathsf{glue}[i = \mathbb{0}_{\mathbb{2}} \mapsto a, i = \mathbb{1}_{\mathbb{2}} \mapsto b]b$ to $\mathsf{glue}[i = \mathbb{0}_{\mathbb{2}} \mapsto \eta_A\ a, i = \mathbb{1}_{\mathbb{2}} \mapsto \eta_B\ b](\eta_B\ b)$. It is sufficient to show that this map is an equivalence. To do so, we construct a homotopy inverse. First, observe that, as η is a strict natural transformation, for any cobar modal types A and B with homotopy inverses to η given by p_A and p_B respectively, we get that the p functions are weakly natural for any function between A and B. Thus, for any term g in $\mathsf{duahom}'\ (\mathsf{D}A)\ (\mathsf{D}B)\ (\mathsf{Dfun}\ f)\ i$, we get a path from $p_B(\mathsf{unglue}\ g)$ to $f(p_A\ g)$ whenever $i = 0$. Given a term g in we can use such a path between $\mathsf{glue}[i = \mathbb{0}_{\mathbb{2}} \mapsto a, i = \mathbb{1}_{\mathbb{2}} \mapsto b]b$, we define a term $\mathsf{fixb}(g)$ in B by composing $p_B(\mathsf{unglue}\ g)$ with this path to force $\mathsf{fixb}(g)$ to equal $f(p_A\ g)$ when $i = \mathbb{0}_{\mathbb{2}}$. Having done this, we can now define the inverse to η by sending g to $\mathsf{glue}[i = \mathbb{0}_{\mathbb{2}} \mapsto p_A\ g, i = \mathbb{1}_{\mathbb{2}} \mapsto \mathsf{fix}(g)]\mathsf{fix}(g)$. Note that when precomposed with $\eta_{\mathsf{duahom}'\ A\ B\ f\ i}$, the resulting function sends $\mathsf{glue}[i = \mathbb{0}_{\mathbb{2}} \mapsto a, i = \mathbb{1}_{\mathbb{2}} \mapsto b]b$ to $\mathsf{glue}[i = \mathbb{0}_{\mathbb{2}} \mapsto p_A(\eta_A\ a), i = \mathbb{1}_{\mathbb{2}} \mapsto \mathsf{fix}(p_B(\eta_B\ b))]\mathsf{fix}(p_B(\eta_B\ b))$ (where fix corresponds to fixb after unglue reduces to be over a term in $\mathsf{D}B$). Given we already have paths between a and $p_A(\eta_A\ a)$ and between b and $p_B(\eta_B\ b)$, we get a path between $\mathsf{glue}[i = \mathbb{0}_{\mathbb{2}} \mapsto a, i = \mathbb{1}_{\mathbb{2}} \mapsto b]b$ and $\mathsf{glue}[i = \mathbb{0}_{\mathbb{2}} \mapsto p_A(\eta_A\ a), i = \mathbb{1}_{\mathbb{2}} \mapsto \mathsf{fix}(p_B(\eta_B\ b))]\mathsf{fix}(p_B(\eta_B\ b))$. ☒

Finally, we show that all of our universes of cobar modal types are themselves cobar modal. While Coquand, Sattler and Ruch already showed that $\mathsf{U}_{\mathsf{cobar}}$ is cobar modal when defined using an arbitrary lex operator (Lemma 4.2.7, Proposition 11 in [28]), for $\mathsf{U}_{\mathsf{covCobar}}$ and $\mathsf{U}_{\mathsf{innerCobar}}$ we must augment their proof using specific information pertaining to cobar.

The crux of the proof relies on being able to conclude that a family of cobar modal types $A : \mathsf{DU}_{\mathsf{cov}} \to \mathsf{U}_{\mathsf{Kan}}$ is covariant using the fact we know $A \circ \eta_{\mathsf{U}_{\mathsf{cov}}}$ is covariant (and analogously for families $A : \mathsf{DU}_{\mathsf{inner}} \to \mathsf{U}_{\mathsf{Kan}}$). As such, we first prove the following lemma:

Lemma 4.2.20. *Consider a type* Γ *in* $\mathsf{U}_{\mathsf{Kan}}$ *a family* $A : [\![\mathsf{D}\Gamma]\!]_{\mathsf{Kan}} \to \mathsf{U}_{\mathsf{cobar}}$*, a representable* $\Psi = \mathbb{I}^n \times 2^m$ *and cofibration* φ *scoped by* Ψ*. Lastly, consider a predicate* $P : (\Sigma x : \Psi.\varphi\ x) \to \mathsf{U}_{\mathsf{cobar}}$*. Given a witness of type* $\Pi p : ((\Sigma x : \Psi.\varphi\ x) \to [\![\Gamma]\!]_{\mathsf{Kan}}).[\![P(\mathsf{fst} \circ A \circ \eta_\Gamma \circ p)]\!]_{\mathsf{cobar}}$*, we can construct a witness to the predicate* $\Pi p : ((\Sigma x : \Psi.\varphi\ x) \to [\![\mathsf{D}\Gamma]\!]_{\mathsf{Kan}}).[\![P(\mathsf{fst} \circ A \circ p)]\!]_{\mathsf{cobar}}$*.*

Proof. Using Lemma 4.2.13, we get an isomorphism $g : \mathsf{D}((\Sigma x : \Psi.\varphi\ x) \to [\![\Gamma]\!]_{\mathsf{Kan}}) \to ((\Sigma x : \Psi.\varphi\ x) \to [\![\mathsf{D}\Gamma]\!]_{\mathsf{Kan}})$. Applying g gives us an isomorphism

$$\Pi p : ((\Sigma x : \Psi.\varphi\ x) \to [\![\Gamma]\!]_{\mathsf{Kan}}).[\![P(\mathsf{fst} \circ A \circ \eta_\Gamma \circ p)]\!]_{\mathsf{cobar}} \simeq \Pi p : ((\Sigma x : \Psi.\varphi\ x) \to [\![\Gamma]\!]_{\mathsf{Kan}}).[\![P(\mathsf{fst} \circ A \circ g\ (\eta_{(\Sigma x:\Psi.\varphi\ x)\to[\![\Gamma]\!]_{\mathsf{Kan}}}\ p))]\!]_{\mathsf{cobar}}$$

Given the codomain is cobar modal and we know D is a modality from Lemma 4.2.12, we can conlude that $f \mapsto f \circ \eta_{(\Sigma x:\Psi.\varphi\ x)\to[\![\Gamma]\!]_{\mathsf{Kan}}}$ is an equivalence. Applying the equivalence, we get a term in the following type:

$$\Pi p : \mathsf{D}((\Sigma x : \Psi.\varphi\ x) \to [\![\Gamma]\!]_{\mathsf{Kan}}).[\![P(\mathsf{fst} \circ A \circ g\ p)]\!]_{\mathsf{cobar}}$$

Lastly, using the isomorphism g again, we a term in

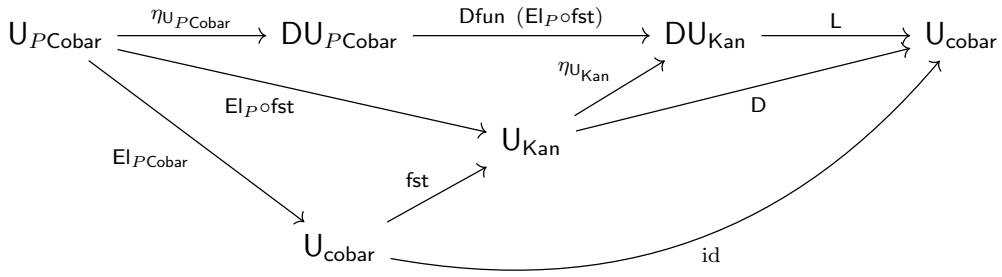
$$\Pi p : ((\Sigma x : \Psi.\varphi\ x) \to [\![\mathsf{D}\Gamma]\!]_{\mathsf{Kan}}).[\![P(\mathsf{fst} \circ A \circ p)]\!]_{\mathsf{cobar}},$$

the witness desired in the lemma statement. ⊠

The proof of the following proposition was developed jointly with Daniel Licata, Nima Rasekh and Robert Rose.

Proposition 4.2.21. *Consider a representable* $\Psi = \mathbb{I}^n \times 2^m$ *and cofibration* φ *scoped by* Ψ *such that* $\Sigma x : \Psi.\varphi\ x$ *is tiny, and consider a predicate* $P : ((\Sigma x : \Psi.\varphi\ x) \to \mathsf{U}_{\mathsf{Kan}}) \to \mathsf{U}_{\mathsf{Kan}}$*. Let* U_P *be the internal universe defined using the LOPS construction with* P *acting as the composition structure and decode function* $\mathsf{El}_P : \mathsf{U}_P \to \mathsf{U}_{\mathsf{Kan}}$*. Assume path univalence holds in the universe* U_P*. The universe* $\mathsf{U}_{P\mathsf{Cobar}} := \Sigma X : \mathsf{U}_P.\mathsf{isModal}\ (\mathsf{El}_P\ X)$ *is cobar modal.*

Proof. Let $\mathsf{El}_{P\mathsf{Cobar}} : \mathsf{U}_{P\mathsf{Cobar}} \to \mathsf{U}_{\mathsf{cobar}}$ be the decode function for $U_{P\mathsf{Cobar}}$ defined as $(A, s) \mapsto (\mathsf{El}_P A, s)$. We now show that the following diagram commutes.



First, $\mathsf{Dfun}\ (\mathsf{El}_P \circ \mathsf{fst}) \circ \eta_{\mathsf{U}_{P\mathsf{Cobar}}} = \eta_{\mathsf{U}_{\mathsf{Kan}}} \circ (\mathsf{El}_P \circ \mathsf{fst})$ by naturality of η. $D = L \circ \eta_{\mathsf{U}_{\mathsf{Kan}}}$ and $\mathsf{fst} \circ \mathsf{El}_{P\mathsf{Cobar}} = \mathsf{El}_P \circ \mathsf{fst}$ both hold by definition. Lastly, $\mathsf{D} \circ \mathsf{fst}$ is an equivalence and thus equal to id up to homotopy as for all types $A : \mathsf{U}_{\mathsf{cobar}}$, we know that A is equivalent to $\mathsf{D}A$ because A is cobar modal.

Now let $p' : \mathsf{DU}_{P\mathsf{Cobar}} \to \mathsf{U}_{\mathsf{cobar}}$ be defined as $L \circ \mathsf{Dfun}(\mathsf{El}_P \circ \mathsf{fst})$. By the outside of the diagram (and postcomposing with fst), we know that $\mathsf{fst} \circ p' \circ \eta_{\mathsf{U}_{P\mathsf{Cobar}}}$ is equal to $\mathsf{fst} \circ \mathsf{El}_{P\mathsf{Cobar}}$ up to homotopy. The latter is definitionally equal to $\mathsf{El}_P \circ \mathsf{fst}$. From the LOPS construction, we know El_P is equipped with a witness for the P filling problem, i.e. a term with type $\Pi p : (\Sigma x : \Psi.\varphi\ x \to \mathsf{U}_P).P(\mathsf{El}_P \circ p)$. Given being cobar modal is a homotopy proposition, this solution lifts to give us a solution for $\mathsf{El}_P \circ \mathsf{fst}$ and given the definitional equality also for $\mathsf{fst} \circ \mathsf{El}_{P\mathsf{Cobar}} : \mathsf{U}_{P\mathsf{Cobar}} \to \mathsf{U}_{\mathsf{Kan}}$. Using the homotopy from the outside of the diagram, we can construct a witness of the P filling problem for $\mathsf{fst} \circ p' \circ \eta_{\mathsf{U}_{P\mathsf{Cobar}}}$. By Lemma 4.2.20, we can remove the η and get a solution for $\mathsf{fst} \circ p'$. Now that we have a solution, we can encode this map so that it lands in the LOPS universe U_P and reattach the witness that the types are cobar modal to define the function $p : \mathsf{DU}_{P\mathsf{Cobar}} \to \mathsf{U}_{P\mathsf{Cobar}}$.

Lastly, we need to show that this map is an inverse for $\eta_{\mathsf{U}_{P\mathsf{Cobar}}}$. From Lemma 4.2.2, it suffices to show that $p \circ \eta_{\mathsf{U}_{P\mathsf{Cobar}}}$ is path equal to the identity. This is the same as showing that, for every $X : \mathsf{U}_{P\mathsf{Cobar}}$, there is a path between X and $p\ (\eta_{\mathsf{U}_{P\mathsf{Cobar}}}\ X)$. Using univalence, this unfolds into providing an equivalence between the types $[\![(\mathsf{El}_P \circ \mathsf{fst})\ X]\!]_{\mathsf{Kan}}$ and$[\![(\mathsf{El}_P \circ \mathsf{fst})\ (p\ (\eta_{\mathsf{U}_{P\mathsf{Cobar}}}\ X))]\!]_{\mathsf{Kan}}$. By the definition of p and the encode-decode rule for LOPS universes,

$[\![(\mathsf{El}_P \circ \mathsf{fst})\ (p\ (\eta_{\mathsf{U}_{P\mathsf{Cobar}}}\ X))]\!]_{\mathsf{Kan}}$ is equal to $[\![\mathsf{fst}\ (p'\ (\eta_{\mathsf{U}_{P\mathsf{Cobar}}}\ X))]\!]_{\mathsf{Kan}}$. We also know by definition that $\mathsf{El}_P \circ \mathsf{fst} = \mathsf{fst} \circ \mathsf{El}_{P\mathsf{Cobar}}$. Thus, we can rewrite the question as instead providing an equivalence between $[\![(\mathsf{fst} \circ \mathsf{El}_{P\mathsf{Cobar}})\ X]\!]_{\mathsf{Kan}}$ and $[\![(\mathsf{fst} \circ p' \circ \eta_{\mathsf{U}_{P\mathsf{Cobar}}})\ X]\!]_{\mathsf{Kan}}$. The diagram at the beginning of this proof constructs a homotopy between $p' \circ \eta_{\mathsf{U}_{P\mathsf{Cobar}}}$ and $\mathsf{El}_{P\mathsf{Cobar}}$, and thus postcomposing by fst and applying the functions to X gives us the equivalence needed to complete the proof. ⊠

Corollary 4.2.22. $\mathsf{U}_{\mathsf{covCobar}}$ *has a code in* $\mathsf{U}_{\mathsf{cobar}}$.

Corollary 4.2.23. $\mathsf{U}_{\mathsf{innerCobar}}$ *has a code in* $\mathsf{U}_{\mathsf{cobar}}$.

4.2.4 Completing Directed Univalence

Next, we show that $\mathsf{U}_{\mathsf{covCobar}}$ satisfies directed univalence.

First, as $\mathsf{U}_{\mathsf{covCobar}}$ is the restriction of $\mathsf{U}_{\mathsf{cov}}$ by a homotopy proposition $\mathsf{isModal}$, the proof of the retraction in $\mathsf{U}_{\mathsf{cov}}$ lifts into $\mathsf{U}_{\mathsf{covCobar}}$ (for more details, see the file `DirUnivalenceReflectionStack.agda` in the formalization). To complete the equivalence, it suffices to show that $\mathsf{dua}\eta\mathsf{fun}$ is an equivalence; then, by function extensionality and path-univalence for $\mathsf{U}_{\mathsf{covCobar}}$, we get a path in $\mathbb{2} \to \mathsf{U}_{\mathsf{covCobar}}$. To construct this, we will externally justify the following axiom:

Definition 4.2.24 (Covariant Equivalence Axiom)**.**

$$\begin{aligned}\mathsf{covEquivAx} : \ & \Pi p\ q{:}\mathbb{2} \to \mathsf{U}_{\mathsf{covCobar}}. \\ & \Pi f{:}(\Pi i : \mathbb{2}.[\![p\ i]\!]_{\mathsf{covcobar}} \to [\![q\ i]\!]_{\mathsf{covcobar}}). \\ & \Pi e_0{:}\mathsf{isEquiv}\ (f\ \mathbb{0}_2).\Pi e_1{:}\mathsf{isEquiv}\ (f\ \mathbb{1}_2). \\ & \Pi i{:}\mathbb{2}.\mathsf{isEquiv}\ (f\ i)\end{aligned}$$

which says that any map between covariant fibrations that is an equivalence at $\mathbb{0}_2$ and at $\mathbb{1}_2$ is an equivalence. Before justifying the axiom, we use it to complete the internal directed univalence equivalence.

Lemma 4.2.25 (`DirUnivalence.dua`η)**.** *For every pair of types* $A\ B : \mathsf{U}_{\mathsf{covCobar}}$ *and every morphism* $p : \mathsf{Hom}_{\mathsf{U}_{\mathsf{covCobar}}}(A, B)$*, the map* $\mathsf{dua}\eta\mathsf{fun}$ *defined in Lemma 4.1.4 is an equivalence,*

so by path-univalence for $\mathsf{U}_{\mathsf{covCobar}}$ *we obtain*

$$\mathsf{dua}\eta : \mathsf{Path}_{\mathsf{Hom}_{\mathsf{U}_{\mathsf{covCobar}}}(A,B)}\,(p, \mathsf{duahom}\ A\ B\ (\mathsf{dcoe}\ A\ B\ p))$$

Proof. To build this path, we use $\mathsf{covEquivAx}$ with $\mathsf{dua}\eta\mathsf{fun}$, as on both $\mathbb{0}_2$ and $\mathbb{1}_2$ $\mathsf{dua}\eta\mathsf{fun}$ is strictly equal to the identity function and thus is an equivalence. We will denote the proofs that it is an equivalence on the endpoints by e_0 and e_1 respectively. As the paths at the endpoints are induced by the identity functions, we can also construct paths between the the paths induced by $\mathsf{dua}\eta\mathsf{fun}$ at $\mathbb{0}_2$ and $\mathbb{1}_2$ and the identity paths at A and B, which we will denote by p_0 and p_1. We now construct the path $\mathsf{dua}\eta$ as follows:

$$\begin{aligned}\lambda i\ j.(\mathsf{com}^{\mathbb{0}_\mathbb{I}\mapsto\mathbb{1}_\mathbb{I}}(\lambda\_.&\mathsf{Path}_{\mathsf{U}_{\mathsf{covCobar}}}\,(p\ j, \mathsf{duahom}\ A\ B\ (\mathsf{dcoe}\ A\ B\ p)\ j))\\ &[j = \mathbb{0}_2 \mapsto \lambda i.p_0\ i,\\ &\ j = \mathbb{1}_2 \mapsto \lambda i.p_1\ i]\\ &\ \mathsf{ua}\ (\mathsf{covEquivAx}\ \_\ \_\ \mathsf{dua}\eta\mathsf{fun}\ e_0\ e_1\ j))\ i\end{aligned}$$

⊠

Theorem 4.2.26 (`DirUnivalence.dua`)**.** *For all* $A\ B : \mathsf{U}_{\mathsf{covCobar}}$*, we have an equivalence*

$$\mathsf{dua} : \mathsf{Equiv}\ (\llbracket A \rrbracket_{\mathsf{covcobar}} \to \llbracket B \rrbracket_{\mathsf{covcobar}})\ (\mathsf{Hom}_{\mathsf{U}_{\mathsf{covCobar}}}\,(A, B))$$

Proof. As we have functions between $\llbracket A \rrbracket_{\mathsf{covcobar}} \to \llbracket B \rrbracket_{\mathsf{covcobar}}$ and $\mathsf{Hom}_{\mathsf{U}_{\mathsf{covCobar}}}\,(A, B)$ that are inverses up to homotopy, we have a quasi-equivalence and thus an equivalence. ⊠

Corollary 4.2.27. *The universe* $\mathsf{U}_{\mathsf{covCobar}}$ *is Rezk (and thus also Segal).*

Proof. To prove this, we use Proposition 8.13 from [62] which allows us to conclude that $\mathsf{U}_{\mathsf{covCobar}}$ is Segal if $\lambda X : \mathsf{U}_{\mathsf{covCobar}}.\mathsf{Hom}_{\mathsf{U}_{\mathsf{covCobar}}}\,(A, X)$ is a covariant fibration for all A in $\mathsf{U}_{\mathsf{covCobar}}$. By directed univalence, this fibration is equivalent to $\lambda X : \mathsf{U}_{\mathsf{covCobar}}.A \to X$ which indeed is a covariant fibration given it is a function space with a fixed closed domain. Thus, we can conclude that $\mathsf{U}_{\mathsf{covCobar}}$ is Segal.

To conclude the universe $\mathsf{U}_{\mathsf{covCobar}}$ is Rezk, we simple must conclude that isomorphisms in the directed path structure coincide with the paths from the homotopical structure. Using directed univalence, we know the isomorphisms of directed paths are equivalent to pairs of invertible functions, i.e. quasi-equivalences. By path univalence, we know that the homotopical paths in $\mathsf{U}_{\mathsf{covCobar}}$ are equivalent to equivalences. As $\mathsf{U}_{\mathsf{covCobar}}$ is a Kan type, quasi-equivalences are equivalent to equivalences, and thus $\mathsf{U}_{\mathsf{covCobar}}$ is Rezk. ⊠

Putting all of these results together, we conclude:

Theorem 4.2.28 (Main Theorem)**.** *There exists a constructive model of a type theory in bicubical sets with a universe of fibrant types ($\mathsf{U}_{\mathsf{cobar}}$), a universe of covariant fibrations ($\mathsf{U}_{\mathsf{covCobar}}$) and a universe of inner fibration ($\mathsf{U}_{\mathsf{innerCobar}}$) such that:*

- $\mathsf{U}_{\mathsf{covCobar}}$ *has a decode function into* $\mathsf{U}_{\mathsf{innerCobar}}$ *and* $\mathsf{U}_{\mathsf{cobar}}$*;*
- $\mathsf{U}_{\mathsf{innerCobar}}$ *has a decode function into* $\mathsf{U}_{\mathsf{cobar}}$*;*
- $\mathsf{U}_{\mathsf{cobar}}$ *is closed under* Π*,* Σ*,* DPath*,* DHom *and contains codes for (smaller)* $\mathsf{U}_{\mathsf{cobar}}$*,* $\mathsf{U}_{\mathsf{covCobar}}$ *and* $\mathsf{U}_{\mathsf{innerCobar}}$*;*
- $\mathsf{U}_{\mathsf{covCobar}}$ *is closed under* Π *(with a fixed closed domain),* Σ*,* DPath *and* DHom*;*
- $\mathsf{U}_{\mathsf{innerCobar}}$ *is closed under* Π *(with a fixed closed domain),* Σ*,* DPath *and* DHom*;*
- $\mathsf{U}_{\mathsf{cobar}}$*,* $\mathsf{U}_{\mathsf{covCobar}}$ *and* $\mathsf{U}_{\mathsf{innerCobar}}$ *are all path univalent;*
- $\mathsf{U}_{\mathsf{covCobar}}$ *is morphism (directed path) univalent and Rezk.*

The above statements are summarized in Figure 4.4.

$$\frac{}{\cdot \vdash \mathsf{U}_{\mathsf{cobar}}\ \mathsf{Type}} \qquad \frac{}{\cdot \vdash \mathsf{U}_{\mathsf{covCobar}}\ \mathsf{Type}} \qquad \frac{}{\cdot \vdash \mathsf{U}_{\mathsf{innerCobar}}\ \mathsf{Type}}$$

$$\frac{}{\cdot \vdash \mathsf{El}^{\mathsf{covCobar}}_{\mathsf{innerCobar}} : \mathsf{U}_{\mathsf{covCobar}} \to \mathsf{U}_{\mathsf{innerCobar}}} \qquad \frac{}{\cdot \vdash \mathsf{El}^{\mathsf{covCobar}}_{\mathsf{cobar}} : \mathsf{U}_{\mathsf{covCobar}} \to \mathsf{U}_{\mathsf{cobar}}}$$

$$\frac{}{\cdot \vdash \mathsf{El}^{\mathsf{innerCobar}}_{\mathsf{cobar}} : \mathsf{U}_{\mathsf{innerCobar}} \to \mathsf{U}_{\mathsf{cobar}}}$$

$$\frac{\Gamma \vdash A : \mathsf{U}_{\mathsf{cobar}} \quad \Gamma, x : A \vdash B\ x : \mathsf{U}_{\mathsf{cobar}}}{\Gamma \vdash \Pi\mathsf{code}(x : A, B\ x) : \mathsf{U}_{\mathsf{cobar}}}$$

$$[\![\Pi\mathsf{code}(x : A, B\ x)]\!]_{\mathsf{cobar}} \equiv \Pi x : A.B$$

$$\frac{\Gamma \vdash A : \mathsf{U}_{\mathsf{cobar}} \quad \Gamma, x : A \vdash B\ x : \mathsf{U}_{\mathsf{cobar}}}{\Gamma \vdash \Sigma\mathsf{code}(x : A, B\ x) : \mathsf{U}_{\mathsf{cobar}}}$$

$$[\![\Sigma\mathsf{code}(x : A, B\ x)]\!]_{\mathsf{cobar}} \equiv \Sigma x : A.B$$

$$\frac{\begin{array}{c}\Gamma, x : \mathbb{I} \vdash A\ x : \mathsf{U}_{\mathsf{cobar}} \\ \Gamma \vdash a_0 : [\![A\ \mathbb{0}_{\mathbb{I}}]\!]_{\mathsf{cobar}} \quad \Gamma \vdash a_1 : [\![A\ \mathbb{1}_{\mathbb{I}}]\!]_{\mathsf{cobar}}\end{array}}{\Gamma \vdash \mathsf{DPathcode}(\lambda x.A, a_0, a_1) : \mathsf{U}_{\mathsf{cobar}}}$$

$$[\![\mathsf{DPathcode}(\lambda x.A, a_0, a_1)]\!]_{\mathsf{cobar}} \equiv \mathsf{DPath}_{\lambda x.A}\ (a_0, a_1)$$

$$\frac{\begin{array}{c}\Gamma, x : \mathbb{2} \vdash A\ x : \mathsf{U}_{\mathsf{cobar}} \\ \Gamma \vdash a_0 : [\![A\ \mathbb{0}_{\mathbb{2}}]\!]_{\mathsf{cobar}} \quad \Gamma \vdash a_1 : [\![A\ \mathbb{1}_{\mathbb{2}}]\!]_{\mathsf{cobar}}\end{array}}{\Gamma \vdash \mathsf{DHomcode}(\lambda x.A, a_0, a_1) : \mathsf{U}_{\mathsf{cobar}}}$$

$$[\![\mathsf{DHomcode}(\lambda x.A, a_0, a_1)]\!]_{\mathsf{cobar}} \equiv \mathsf{DHom}_{\lambda x.A}\ (a_0, a_1)$$

Figure 4.4: Closure conditions and properties of the various universes (part 1)

$$\frac{i < j}{\cdot \vdash \mathsf{UCobarcode}^i : \mathsf{U}^{\mathsf{j}}_{\mathsf{cobar}}} \qquad [\![\mathsf{UCobarcode}^i]\!]_{\mathsf{cobar}} \equiv \mathsf{U}^{\mathsf{i}}_{\mathsf{cobar}}$$

$$\frac{i < j}{\cdot \vdash \mathsf{UCovCobarcode}^i : \mathsf{U}^{\mathsf{j}}_{\mathsf{cobar}}} \qquad [\![\mathsf{UCovCobarcode}^i]\!]_{\mathsf{cobar}} \equiv \mathsf{U}^{\mathsf{i}}_{\mathsf{covCobar}}$$

$$\frac{i < j}{\cdot \vdash \mathsf{UInnerCobarcode}^i : \mathsf{U}^{\mathsf{j}}_{\mathsf{cobar}}} \qquad [\![\mathsf{UInnerCobarcode}^i]\!]_{\mathsf{cobar}} \equiv \mathsf{U}^{\mathsf{i}}_{\mathsf{innerCobar}}$$

$$\frac{\cdot \vdash A : \mathsf{U}_{\mathsf{covCobar}} \quad \Gamma, x : A \vdash B\ x : \mathsf{U}_{\mathsf{covCobar}}}{\Gamma \vdash \mathsf{\Pi code}_{\mathsf{cov}}(x : A, B\ x) : \mathsf{U}_{\mathsf{covCobar}}}$$

$$[\![\mathsf{\Pi code}_{\mathsf{cov}}(x : A, B\ x)]\!]_{\mathsf{covcobar}} \equiv \Pi x : A.B$$

$$\frac{\Gamma \vdash A : \mathsf{U}_{\mathsf{covCobar}} \quad \Gamma, x : A \vdash B\ x : \mathsf{U}_{\mathsf{covCobar}}}{\Gamma \vdash \mathsf{\Sigma code}_{\mathsf{cov}}(x : A, B\ x) : \mathsf{U}_{\mathsf{covCobar}}}$$

$$[\![\mathsf{\Sigma code}_{\mathsf{cov}}(x : A, B\ x)]\!]_{\mathsf{covcobar}} \equiv \Sigma x : A.B$$

$$\frac{\begin{array}{c}\Gamma, x : \mathbb{I} \vdash A\ x : \mathsf{U}_{\mathsf{covCobar}} \\ \Gamma \vdash a_0 : [\![A\ \mathbb{0}_{\mathbb{I}}]\!]_{\mathsf{covcobar}} \quad \Gamma \vdash a_1 : [\![A\ \mathbb{1}_{\mathbb{I}}]\!]_{\mathsf{covcobar}}\end{array}}{\Gamma \vdash \mathsf{DPathcode}_{\mathsf{cov}}(\lambda x.A, a_0, a_1) : \mathsf{U}_{\mathsf{covCobar}}}$$

$$[\![\mathsf{DPathcode}_{\mathsf{cov}}(\lambda x.A, a_0, a_1)]\!]_{\mathsf{covcobar}} \equiv \mathsf{DPath}_{\lambda x.A}\,(a_0, a_1)$$

$$\frac{\begin{array}{c}\Gamma, x : \mathbb{2} \vdash A\ x : \mathsf{U}_{\mathsf{covCobar}} \\ \Gamma \vdash a_0 : [\![A\ \mathbb{0}_{\mathbb{2}}]\!]_{\mathsf{covcobar}} \quad \Gamma \vdash a_1 : [\![A\ \mathbb{1}_{\mathbb{2}}]\!]_{\mathsf{covcobar}}\end{array}}{\Gamma \vdash \mathsf{DHomcode}_{\mathsf{cov}}(\lambda x.A, a_0, a_1) : \mathsf{U}_{\mathsf{covCobar}}}$$

$$[\![\mathsf{DHomcode}_{\mathsf{inner}}(\lambda x.A, a_0, a_1)]\!]_{\mathsf{innercobar}} \equiv \mathsf{DHom}_{\lambda x.A}\,(a_0, a_1)$$

Figure 4.4: Closure conditions and properties of the various universes (part 2)

$$\frac{\cdot \vdash A : \mathsf{U}_{\mathsf{innerCobar}} \quad \Gamma, x : A \vdash B\ x : \mathsf{U}_{\mathsf{innerCobar}}}{\Gamma \vdash \Pi\mathsf{code}_{\mathsf{inner}}(x : A, B\ x) : \mathsf{U}_{\mathsf{innerCobar}}}$$

$$[\![\Pi\mathsf{code}_{\mathsf{inner}}(x : A, B\ x)]\!]_{\mathsf{innercobar}} \equiv \Pi x : A.B$$

$$\frac{\Gamma \vdash A : \mathsf{U}_{\mathsf{innerCobar}} \quad \Gamma, x : A \vdash B\ x : \mathsf{U}_{\mathsf{innerCobar}}}{\Gamma \vdash \Sigma\mathsf{code}_{\mathsf{inner}}(x : A, B\ x) : \mathsf{U}_{\mathsf{innerCobar}}}$$

$$[\![\Sigma\mathsf{code}_{\mathsf{inner}}(x : A, B\ x)]\!]_{\mathsf{innercobar}} \equiv \Sigma x : A.B$$

$$\frac{\begin{array}{c}\Gamma, x : \mathbb{I} \vdash A\ x : \mathsf{U}_{\mathsf{innerCobar}} \\ \Gamma \vdash a_0 : [\![A\ \mathbb{0}_{\mathbb{I}}]\!]_{\mathsf{innercobar}} \quad \Gamma \vdash a_1 : [\![A\ \mathbb{1}_{\mathbb{I}}]\!]_{\mathsf{innercobar}}\end{array}}{\Gamma \vdash \mathsf{DPathcode}_{\mathsf{inner}}(\lambda x.A, a_0, a_1) : \mathsf{U}_{\mathsf{innerCobar}}}$$

$$[\![\mathsf{DPathcode}_{\mathsf{inner}}(\lambda x.A, a_0, a_1)]\!]_{\mathsf{innercobar}} \equiv \mathsf{DPath}_{\lambda x.A}\ (a_0, a_1)$$

$$\frac{\begin{array}{c}\Gamma, x : \mathbb{2} \vdash A\ x : \mathsf{U}_{\mathsf{innerCobar}} \\ \Gamma \vdash a_0 : [\![A\ \mathbb{0}_{\mathbb{2}}]\!]_{\mathsf{innercobar}} \quad \Gamma \vdash a_1 : [\![A\ \mathbb{1}_{\mathbb{2}}]\!]_{\mathsf{innercobar}}\end{array}}{\Gamma \vdash \mathsf{DHomcode}_{\mathsf{inner}}(\lambda x.A, a_0, a_1) : \mathsf{U}_{\mathsf{innerCobar}}}$$

$$[\![\mathsf{DHomcode}_{\mathsf{inner}}(\lambda x.A, a_0, a_1)]\!]_{\mathsf{innercobar}} \equiv \mathsf{DHom}_{\lambda x.A}\ (a_0, a_1)$$

$$\frac{}{\cdot \vdash \mathsf{ua} : \Pi A\ B : \mathsf{U}_{\mathsf{cobar}}.\mathsf{Equiv}\ (\mathsf{Equiv}\ A\ B)\ (\mathsf{Path}_{\mathsf{U}_{\mathsf{cobar}}}\ (A, B))}$$

$$\frac{}{\cdot \vdash \mathsf{ua}_{\mathsf{cov}} : \Pi A\ B : \mathsf{U}_{\mathsf{covCobar}}.\mathsf{Equiv}\ (\mathsf{Equiv}\ A\ B)\ (\mathsf{Path}_{\mathsf{U}_{\mathsf{covCobar}}}\ (A, B))}$$

$$\frac{}{\cdot \vdash \mathsf{ua}_{\mathsf{inner}} : \Pi A\ B : \mathsf{U}_{\mathsf{innerCobar}}.\mathsf{Equiv}\ (\mathsf{Equiv}\ A\ B)\ (\mathsf{Path}_{\mathsf{U}_{\mathsf{innerCobar}}}\ (A, B))}$$

$$\frac{}{\cdot \vdash \mathsf{dua} : \Pi A\ B : \mathsf{U}_{\mathsf{covCobar}}.\mathsf{Equiv}\ (A \to B)\ (\mathsf{Hom}_{\mathsf{U}_{\mathsf{covCobar}}}\ (A, B))}$$

Figure 4.4: Closure conditions and properties of the various universes (part 3)

4.2.5 The Equivalence Axiom in Bicubical Sets

Having used the equivalence axiom to complete our main theorem, we now provide an external constructive proof justifying its inclusion in the model. While we do not define a model structure on bicubical sets, we do use some aspects of the type-theoretic model structure on cubical sets (see [12, 29]) in the following definitions and proofs.

The main theorem we will use that leverages the added structure from cobar is as follows:

Theorem 4.2.29 (Theorem in Section 5.2 in [28])**.** *Given cobar modal types A and B in $\mathsf{U}_{\mathsf{Kan}}$ and a function $f : [\![A]\!]_{\mathsf{Kan}} \to [\![B]\!]_{\mathsf{Kan}}$, if for every $X \in \square_{\mathrm{Ded}}$ $f(-, X)$ is an equivalence of cubical sets, then f is an equivalence of bicubical sets.*

Now, we begin the proof:

Definition 4.2.30. An object $X \in \mathbf{Set}^{\square^{\mathrm{op}}_{\mathrm{path}}}$ is the *homotopy pullback* of morphisms f from A to C and g from B to C if it is equipped with morphisms to A and B such that the following square satisfies the standard pullback universal property up to homotopy.

$$\begin{array}{ccc} X & \longrightarrow & B \\ \downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & C \end{array}$$

That is, all of the required equations on morphisms are true up to the notion of path equality in the type theory on cubical sets, as is the object only unique up to path equality/homotopy.

Definition 4.2.31. A morphism f in the category $\mathbf{Set}^{\square^{\mathrm{op}}_{\mathrm{path}} \times \square^{\mathrm{op}}_{\mathrm{Ded}}}$ is a *left map* if, for every $n \in \mathbb{N}$, the commuting square of cubical sets shown below is a homotopy pullback.

$$\begin{array}{ccc} A(\_, n) & \xrightarrow{\mathbb{0}_2} & A(\_, 0) \\ {\scriptstyle f(\_,n)}\downarrow & & \downarrow{\scriptstyle f(\_,0)} \\ \Gamma(\_, n) & \xrightarrow[\mathbb{0}_2]{} & \Gamma(\_, 0) \end{array}$$

The map denoted by $\mathbb{0}_2$ is that induced by the substitution sending all directed interval variables to $\mathbb{0}_2$. A morphism is a *left fibration* if it is both a left map and a Kan fibration.

This definition unfolds to the equivalent statement that, for any directed n-cube in Γ and any point in A over its zero vertex, one can construct a unique (up to homotopy) directed n-cube in A over that in Γ.

The internal definition of $\mathsf{covFill}$ for a type family $A : \Gamma \to \mathsf{Type}$ picks out the same morphisms of bicubical sets.

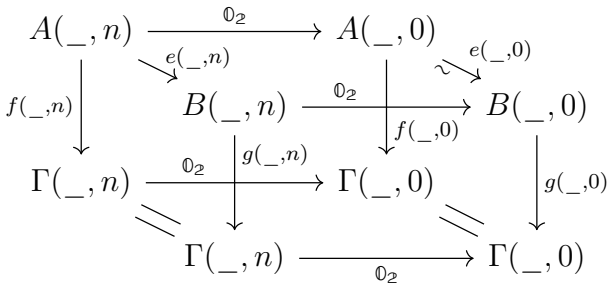
Lemma 4.2.32. *Given some internal type family $B : A \to \mathsf{U}_{\mathsf{cov}}$ in an open context Γ (i.e. $\Gamma \vdash B : A \to \mathsf{U}_{\mathsf{cov}}$), the corresponding external morphism $\Sigma x : \Gamma.\Sigma y : A\ x.B\ x\ y \to \Sigma x : \Gamma.A\ x$ is a left map.*

Proof. As described in Definition 3.2.9, the definition of $\mathsf{covFill}$ picks out type families for which, given any morphism p in Γ and term in A over p at $\mathbb{0}_2$, there is a unique morphism in A over p; however, as any instance of $\mathsf{covFill}$ can be used in an open context and the category $\square_{\mathrm{Ded}}$ is the free Cartesian category generated from 2, the instance can be used n many times in a context with (at least) n many directed interval variables to construct the solution for the directed n-cube. Thus, the external morphism is a left map. ⊠

The covariant equivalence axiom is a direct corollary of a general lemma from [68]. As with the definition of left map, the lemma originally was stated for the Reedy model structure on bisimplicial sets. The proof itself only uses the definition of left map as a level wise homotopy pullback over the zero cells (Lemma 4.2.32), and the fact that equivalences are level wise equivalences (Theorem 4.2.29); thus, it also holds in our setting. The lemma is as follows.

Lemma 4.2.33. *Given bicubical sets Γ, A and B, two left fibrations $f : A \to \Gamma$ and $g : B \to \Gamma$, and a morphism $e : A \to B$ such that $f = g \circ e$, the morphism e is an equivalence of bicubical sets when $e(\_, 0) : A(\_, 0) \to B(\_, 0)$ is an equivalence of cubical sets.*

Proof. For each $n \in \mathbb{N}$ consider the following diagram.



As both the front and back squares are homotopy pullback diagrams and $e(\_,0)$ is an equivalence of cubical sets, it follows that $e(\_,n)$ is also an equivalence of cubical sets. As equivalences are level wise equivalences of cubical sets, e is an equivalence of bicubical sets. ⊠

Corollary 4.2.34. *Consider bicubical sets Γ, A and B, two left fibrations $f : A \to \Gamma \times \mathbb{2}$ and $g : B \to \Gamma \times \mathbb{2}$, and a morphism $e : A \to B$ such that $(f,p) = (g,q) \circ e$. Let $A_{\mathbb{2}(0)}$ denote the homotopy pullback of $f(\_,0)$ and the $\mathbb{0}_2$-projection $\Gamma \times \mathbb{2}(\_,0) \hookrightarrow \Gamma(\_,0) \times \mathbb{2}(\_,0)$ (and define $B_{\mathbb{2}(0)}$ analogously) as shown in the following diagram.*

$$\begin{array}{ccc} A_{\mathbb{2}(0)} & \xrightarrow{\mathbb{0}_2} & A(\_,0) \\ {\scriptstyle f_{\mathbb{2}(0)}}\downarrow & & \downarrow{\scriptstyle f(\_,0)} \\ \Gamma \times \mathbb{2}(\_,0) & \xrightarrow[\mathbb{0}_2]{} & \Gamma(\_,0) \times \mathbb{2}(\_,0) \end{array}$$

Let $e_{\mathbb{2}(0)} : A_{\mathbb{2}(0)} \to B_{\mathbb{2}(0)}$ be the morphism induced by e. If $e_{\mathbb{2}(0)}$ is an equivalence of bicubical sets, then e is also an equivalence of bicubical sets.

Proof. As $e_{\mathbb{2}(0)}$ is an equivalence of bicubical sets, it is a level wise equivalence of cubical sets. In particular, $e_{\mathbb{2}(0)}(\_,0) : A_{\mathbb{2}(0)}(\_,0) \to B_{\mathbb{2}(0)}(\_,0)$ is an equivalence. Observe that $A_{\mathbb{2}(0)}(\_,0) = A(\_,0)$ (and similarly for B), and $e_{\mathbb{2}(0)}(\_,0) = e(\_,0)$. As $e(\_,0)$ is an equivalence of cubical sets, Lemma 4.2.33 allows us to conclude that e is an equivalence of bicubical sets. ⊠

Theorem 4.2.35. covEquivAx *holds in our model.*

Proof. In order to see Corollary 4.2.34 as a justification of the covariant equivalence axiom (Definition 4.2.24), we must make a couple observations. First, from Lemma 4.2.32, we know that the internal term $\Gamma \vdash p : 2 \to \mathsf{U}_{\mathsf{cov}}$ corresponds to the external left fibration $\Sigma x : \Gamma.\Sigma i : 2.p\ x\ i \to \Gamma \times 2$ (and analogously for q). Thus, in the external corollary, we can instantiate A and B with the external closed bicubical sets $\Sigma x : \Gamma.\Sigma i : 2.p\ x\ i$ and $\Sigma x : \Gamma.\Sigma i : 2.q\ x\ i$. Let A and B denote these bicubical sets in the following proof. The second item of note is that the pullback defining $A_{2(0)}$ in the statement of the corollary corresponds to the restriction of $A \to \Gamma \times 2$ to the components over the two endpoints $\mathbb{0}_2$ and $\mathbb{1}_2$ of 2. Thus, the internal proofs e_0 and e_1 in the open context Γ demonstrating f is an equivalence over the endpoints of the directed interval matches up with the equivalence needed in the statement of Corollary 4.2.34.

We also need to confirm that the proof of $\mathsf{covEquivAx}$ is stable under substitution, and thus can be an internal axiom. Given $\mathsf{covEquivAx}$ uses Theorem 4.2.29 to construct the equivalence from the input data, we first show that for any A, B and $f : A \to B$ where f is a point-wise equivalence, the term of $\mathsf{isEquiv}\ f$ built using Theorem 4.2.29 is stable under (strict) pullbacks. As the construction in Theorem 4.2.29 described in Theorem 5.1 of [28] is computed point-wise with respect to $\square_{\mathrm{Ded}}$ and pullbacks of bicubical sets are also point-wise pullbacks of cubical sets, the pullback commutes with the construction and thus Theorem 4.2.29 commutes with substitution. The second construction that must be stable under pullbacks is the level-wise equivalence defined in Lemma 4.2.33. In particular, given cubical sets Γ, A and B, left fibrations $f : A \to \Gamma$ and $g : B \to \Gamma$, a morphism $e : A \to B$ such that $f = g \circ e$, and a proof $v_0 : \mathsf{isEquiv}\ e(\_, 0)$, we need to know that every $v_n : \mathsf{isEquiv}\ e(\_, n)$ created by the proof is stable under pullback. Again, as each v_n is computed separately for each object in $\square_{\mathrm{Ded}}$, the pullback commutes with the point-wise argument, and as every v_n is constructed as the solution of a covariant filling problem (which in particular commutes with pullbacks), the construction in Lemma 4.2.33 is also stable under substitution. ⊠

Chapter 5

Fiberwise Fibrancy and Internal Universes

The contents of this chapter are joint work with Daniel Licata and were presented at the 2nd International Conference on Homotopy Type Theory [82].

One of the fundamental characteristics that distinguished homotopy type theory from other type theories is that all equational reasoning is done up to homotopy, and strict equality is inherently inaccessible and incompatible with the type theory; to immediately contradict this statement, there are a few isolated situations where we do safely interact with strict equality, namely when describing boundary constraints of types. The first place one is likely to encounter such a strict equality is when working with composition structures on types: the composition term $\mathsf{com}_A^{z:\mathbb{0}_\mathbb{I}\to\mathbb{1}_\mathbb{I}}\ [\alpha \mapsto t\ z]\ b$ is by definition strictly equal to $t\ \mathbb{1}_\mathbb{I}$ whenever the cofibration α holds true. The second common safe occurrence of strict equality is in RS extension types (Definition 2.3.11): Generalizing path (and morphism) types, RS extension types are Π-types of the form $\Pi\vec{x} : \mathbb{I}^n.A[\alpha \mapsto a]$ such that α is scoped entirely by the variables $\vec{x}$. Given a term f of the RS extension type above, we know $f\ x$ is strictly equal to a whenever α is true. By restricting α to only use variables bound in the Π, the resulting type remains fibrant despite introducing and enforcing a strict equality on the terms inhabiting it.

Generalizing from strict equality, strict propositions as a whole are very limited in their use within homotopy type theory, in particular only appearing as the cofibrations in the above instances. In this chapter, we will develop a theory that greatly generalizes when it is safe to use both cofibration witness types (ultimately defining partial types)—$[\![\alpha]\!]_{\mathsf{cof}}$—and boundary types (also commonly called extension types, Definition 2.3.4)—$A[\alpha \mapsto a]$—and after doing so explore an application made possible by this theory: a syntax for internally-defined LOPS universes.

First, let us see what is possible without the theory defined in this chapter, and demonstrate why the new mechanisms introduced here are necessary. The final commonly used type that is built using both cofibration witness types and a boundary type is the contractible fill type (Definition 2.3.12).

$$\begin{aligned} \mathsf{cfill}\ A := \quad & \Pi \alpha : \mathsf{Cof}. \\ & \Pi a_\alpha : [\![\alpha]\!]_{\mathsf{cof}} \to A. \\ & A[\alpha \mapsto a_\alpha] \end{aligned}$$

An equivalent notion to standard contractibility (assuming the type A is Kan), this predicate on a type A allows one to fill any partial term a_α to a total term that strictly agrees with a_α when the cofibration α is satisfied. Combining RS extension types with the contractible fill type, one can already define a great deal of interest. In the case of our application, we wish to define the types of filling problem predicates and ensure they are fibrant; notably, many can be built using these two constructions alone. First, let us consider one of the potential filling problem one can use to define when a type in DeMorgan cubical type theory is fibrant (which also is the filling problem needed for simplicial homotopy type theory, and our notion of covariance given in Chapter 3 if you swap the choice of interval):

$$\begin{aligned} \mathsf{comFill}^{\mathbb{I}}\ (A : \mathbb{I} \to \mathsf{U}) := \quad & \Pi \alpha : \mathsf{Cof}. \\ & \Pi b : A\ \mathbb{0}_{\mathbb{I}}. \\ & \Pi t : \Pi z : \mathbb{I}.[\![\alpha]\!]_{\mathsf{cof}} \to A\ z[z = \mathbb{0}_{\mathbb{I}} \mapsto b]. \\ & \Pi z : \mathbb{I}. \to A[z = \mathbb{0}_{\mathbb{I}} \mapsto b, \alpha \mapsto t] \\ \mathsf{comFill}\ \Gamma\ (A : \Gamma \to \mathsf{U}) := \quad & \Pi p : \mathbb{I} \to \Gamma.\mathsf{comFill}^{\mathbb{I}}\ (A \circ p) \end{aligned}$$

We can rewrite the type $\mathsf{comFill}^{\mathbb{I}}\ A$ to an isomorphic type as the following.

$$\mathsf{comFill}^{\mathbb{I}}\ (A : \mathbb{I} \to \mathsf{U}) := \Pi b : A\ \mathbb{0}_{\mathbb{I}}.\mathsf{cfill}\ (\Pi z : \mathbb{I}.A\ z[z = \mathbb{0}_{\mathbb{I}} \mapsto b])$$

From this, it initially seems leveraging this pattern is sufficient for all filling problems. As our second example, let us now consider the fibrancy condition used in Cartesian cubical type theory.

$$\begin{aligned}
\mathsf{comFill}^{\mathbb{I}}_{\mathrm{Cart}}\ (A : \mathbb{I} \to \mathsf{U}) := \quad & \Pi\alpha : \mathsf{Cof}. \\
& \Pi i : \mathbb{I}. \\
& \Pi b : A\ i. \\
& \Pi t : \Pi z : \mathbb{I}.[\![\alpha]\!]_{\mathsf{cof}} \to A\ z[z = i \mapsto b]. \\
& \Pi z : \mathbb{I}. \to A[z = i \mapsto b, \alpha \mapsto t] \\
\mathsf{comFill}_{\mathrm{Cart}}\ \Gamma\ (A : \Gamma \to \mathsf{U}) := \quad & \Pi p : \mathbb{I} \to \Gamma.\mathsf{comFill}^{\mathbb{I}}_{\mathrm{Cart}}\ (A \circ p)
\end{aligned}$$

Like before, we can rewrite this version of $\mathsf{comFill}^{\mathbb{I}}_{\mathrm{Cart}}\ A$ as an isomorphic type using an instance of cfill and (what looks like) an RS extension type.

$$\mathsf{comFill}^{\mathbb{I}}_{\mathrm{Cart}}\ (A : \mathbb{I} \to \mathsf{U}) := \Pi i : \mathbb{I}.\Pi b : A\ i.\mathsf{cfill}\ (\Pi z : \mathbb{I}.A\ z[z = i \mapsto b])$$

While certainly this type is isomorphic to our first definition of $\mathsf{comFill}^{\mathbb{I}}_{\mathrm{Cart}}\ A$, unfortunately the type contained in the contractible fill is not actually an RS extension type; the cofibration depends on the $i : \mathbb{I}$ bound in the outermost Π, and thus is not entirely scoped by the interval variables bound directly before the boundary type. This poses a challenge as the notion of Kan fibrancy corresponds to filling shapes built out of such interval variables, and some types used to build up the body of the Π-type depend on an interval variable that cause them to clearly not be fibrant when considered locally in isolation from the context in which they are used. Another filling problem that faces a similar problem and cannot be described simply with RS extension types and contractible fill is the modified notion of covariance described in Section 6.1, its challenge pertaining to directed interval variables and thus complicating the question of whether the type itself can be equipped with a solution to the covariant filling

problem (which focuses on filling shapes built from the directed interval). Thankfully, both of these type are ultimately still fibrant, and the remainder of this chapter will provide a general theory justifying why it is the case.

5.1 Fiberwise Filling in a Topos

As our first approach do defining this theory, we will work within the internal logic of a topos. Given we encounter many models of type theory defined within presheaf topoi over varying categories (e.g. Cartesian cubical sets, bisimplicial sets), We work in the internal logic of an arbitrary presheaf topos indexed by a category $\mathbb{C}$. We will write U to denote the object classifier and Ω to denote the subobject classifier. We assume the presence of a cofibration predicate on the subobject classifier, $\mathsf{iscof} \in \mathsf{Hom}_{\widehat{\mathbb{C}}}(\Omega, \Omega)$, and write Cof to denote the subobject of Ω classified by iscof. Given a (product of) representables Φ, we write $\Phi \vdash \varphi\ \mathsf{cofib}$ when φ is a morphism in $\mathsf{Hom}_{\widehat{\mathbb{C}}}(\Phi, \mathsf{Cof})$, and we write $[\varphi]$ to denote the object that witnesses when the proposition φ holds true. Lastly, for $\Phi \vdash \varphi\ \mathsf{cofib}$ and $\Phi \vdash \psi\ \mathsf{cofib}$, we write $\varphi \vdash_\Phi \psi$ when the proposition φ entails ψ, or equivalently that $\Phi.[\varphi]$ is itself a subobject of $\Phi.[\psi]$.

5.1.1 Filling Operations

Given we wish to work as abstractly as possible, we not only work within an arbitrary presheaf topos, but we also define an abstract notion of filling operation with which to classify the fibrant types in the topos model. This decision is particularly beneficial to work in directed type theory, given it requires utilizing a large number of filling problems simultaneously within the same type theory. The definition we develop in this section is sufficiently general such that it captures all of the common filling problems used in homotopy and cubical type theories of which we currently are aware. Before we write down the formal definition of a filling operation, let's first unpack a few familiar instances of filling problems and identify

their components from which we can slowly abstract the generalized definition. As related work, note that Nuyts has also explored some generalized notions of filling problems [56]; for a bit more commentary on his work in relation to that presented in this chapter, see Section 1.4.

The first pattern we encounter originates from LOPS [47]: To define a predicate classifying solutions to a filling operation on an arbitrary type A depending on context Γ, we begin by first defining a predicate on types depending solely on the interval $\mathbb{I}$, and then use this predicate to define the one for general contexts Γ by quantifying over all paths $p : \mathbb{I} \to \Gamma$ and composing. More concretely, consider the following version of Kan composition.

$$\begin{aligned}
\mathsf{comFill}^{\mathbb{I}}\ (A : \mathbb{I} \to \mathsf{U}) :=\ & \Pi\alpha : \mathsf{Cof}. \\
& \Pi b : A\ \mathbb{0}_{\mathbb{I}}. \\
& \Pi t : \Pi z : \mathbb{I}.[\![\alpha]\!]_{\mathsf{cof}} \to A\ z[z = \mathbb{0}_{\mathbb{I}} \mapsto b]. \\
& \Pi z : \mathbb{I}. \to A\ z[z = \mathbb{0}_{\mathbb{I}} \mapsto b, \alpha \mapsto t] \\
\mathsf{comFill}\ \Gamma\ (A : \Gamma \to \mathsf{U}) :=\ & \Pi p : \mathbb{I} \to \Gamma.\mathsf{comFill}^{\mathbb{I}}\ (A \circ p)
\end{aligned}$$

The shape of the filling problem is given in the predicate $\mathsf{comFill}^{\mathbb{I}} : (\mathbb{I} \to \mathsf{U}) \to \mathsf{U}$, and using this predicate we define the actual filling operation predicate $\mathsf{comFill} : \Pi\Gamma : \mathsf{U}.(\Gamma \to \mathsf{U}) \to \mathsf{U}$ as $\mathsf{comFill}\ \Gamma\ A := \Pi p : \mathbb{I} \to \Gamma.\mathsf{comFill}^{\mathbb{I}}\ (A \circ p)$. As this pattern appears universally, we shall copy this structure in our definition, and in particular this means the majority of our focus will be on the initial predicate that carries the shape of the filling problem.

The second abstraction we wish to identify also is identified in LOPS [47]: The predicate need not classify all types, but may instead be defined as a predicate on an existing universe. We use this ourselves in defining the predicate for covariant filling (Definition 3.2.9). Specifically, consider the types of the predicates we defined in Section 3.2.3.

$$\begin{aligned}
\mathsf{covFill}^{2} :\ & (2 \to \mathsf{U}_{\mathsf{Kan}}) \to \mathsf{U} \\
\mathsf{covFill}_{\Gamma} :\ & \Pi\Gamma : \mathsf{U}.(2 \to \mathsf{U}_{\mathsf{Kan}}) \to \mathsf{U}
\end{aligned}$$

As ultimately we use LOPS to define $\mathsf{U}_{\mathsf{cov}}$ as a subuniverse of $\mathsf{U}_{\mathsf{Kan}}$, the predicate itself is only defined to classify types in $\mathsf{U}_{\mathsf{Kan}}$ and not arbitrary presheaves in the topos. That being said, we think of contexts as being arbitrary presheaves, and thus the type of Γ is given by

the object classifier U, and the predicate itself also has type U, as (at this point) we have yet to provide any filling operations for it with which we could pair with the type such that it lands in another universe. Thus, in our generalized definition, we allow for the type family to potentially land in a different universe than the object classifier U.

The next pattern of filling operation predicates relates to how one expresses the partial shape from which we fill; to exemplify this, we can slightly rewrite the above definition of $\mathsf{comFill}^{\mathbb{I}}$.

$$\begin{aligned}\mathsf{comFill}^{\mathbb{I}}\ (A : \mathbb{I} \to \mathsf{U}) := \quad & \Pi\alpha : \mathsf{Cof}. \\ & \Pi b : \Pi z : \mathbb{I}.[\![z = \mathbb{0}_{\mathbb{I}}]\!]_{\mathsf{cof}} \to A\ z. \\ & \Pi t : \Pi z : \mathbb{I}.[\![\alpha]\!]_{\mathsf{cof}} \to A\ z[z = \mathbb{0}_{\mathbb{I}} \mapsto b]. \\ & \Pi z : \mathbb{I}. \to A\ z[z = \mathbb{0}_{\mathbb{I}} \mapsto b, \alpha \mapsto t]\end{aligned}$$

We think of filling operations as filling a shape from the base b along the tube t, where the base b is a partial restriction of the entire shape that is ultimately built in the solution of the filling problem, and the tube t is defined along the entirety of the shape being filled, but is restricted to be partial along an arbitrary cofibration. For those familiar with model category theory, this corresponds to the filling operation being a solution to all right lifts of the pushout product of a generating trivial cofibration (given by the restriction on b) with an arbitrary cofibration (the α that restricts t). For those not well-versed in model category theory, the pattern to identify is that b is restricted by some cofibration on the interval $\mathbb{I}$ we are filling, t is always restricted by an arbitrary cofibration and agrees with b along its cofibration, and the output agrees with b and t everywhere they are defined. In the above example, the cofibration restricting b is $z = \mathbb{0}_{\mathbb{I}}$, and thus the "shape" of the filling operation (i.e. the trivial cofibration) is the inclusion sending the point to $\mathbb{0}_{\mathbb{I}}$ in the interval. Abstracting this pattern, one can define the operation that fills the interval $\mathbb{I}$ from any cofibration $z : \mathbb{I} \vdash \delta\,\mathsf{cofib}$ as being that defined using the following.

$$\begin{aligned}\mathsf{Fill}^{\mathbb{I}}\ (A : \mathbb{I} \to \mathsf{U}) := \quad & \Pi\alpha : \mathsf{Cof}. \\ & \Pi b : \Pi z : \mathbb{I}.[\![\delta\ z]\!]_{\mathsf{cof}} \to A\ z. \\ & \Pi t : \Pi z : \mathbb{I}.[\![\alpha]\!]_{\mathsf{cof}} \to A\ z[\delta\ z \mapsto b]. \\ & \Pi z : \mathbb{I}.A\ z[\delta\ z \mapsto b, \alpha \mapsto t]\end{aligned}$$

Another variation we see is the total shape we are filling with the filling operation. To exemplify this scenario, consider the filling operation used do define inner fibrations in Section 3.2.5.

$$\begin{aligned}\mathsf{innerFill}^{\Delta}\ (A : \Delta^2 \to \mathsf{U}) := \quad & \Pi\alpha{:}\Omega_{\mathrm{cof}}. \\ & \Pi t{:}(\Pi z{:}\Delta^2.[\![\alpha]\!]_{\mathsf{cof}} \to A\ z). \\ & \Pi b{:}(\Pi z{:}\Delta^2.[\![\Lambda^2_1\ z]\!]_{\mathsf{cof}} \to A\ z[\alpha \mapsto t\ z]). \\ & \Pi z{:}\Delta^2.A\ z[[\![\Lambda^2_1\ z]\!]_{\mathsf{cof}} \mapsto b\ z, \alpha \mapsto t\ z]\end{aligned}$$

This filling operation does not fill an interval, but instead is filling a triangle, given by $\Sigma(x, y) : 2^2.[\![y \leq x]\!]_{\mathsf{cof}}$. To capture this pattern, we introduce two significant abstractions: first, instead of always filling an interval we are now filling a product of intervals, or to be even more general a product of representables; second, we are not required to fill the entire shape given by the product of representables, but instead allow for a cofibration to restrict to a subshape of the product of representables. To account for this, let Ψ be a product of representables and let $\Psi \vdash \gamma\ \mathsf{cofib}$ be a cofibrations restricting Ψ. The idea now is for the filling operation to fill the shape $\Sigma\Psi.[\![\gamma]\!]_{\mathsf{cof}}$. That being said, we still need to fix a subshape of $\Sigma\Psi.[\![\gamma]\!]_{\mathsf{cof}}$ from which we can fill the entire shape. For this, we pick out another cofibration (as described in the previous paragraph), $\Psi \vdash \delta\ \mathsf{cofib}$ such that δ entails γ holds, which in particular indicates that the shape $\Sigma\Psi.[\![\delta]\!]_{\mathsf{cof}}$ is actually a subshape of $\Sigma\Psi.[\![\gamma]\!]_{\mathsf{cof}}$. Taking this into account, we can write down the classifying predicate for a further generalized notion of filling operation.

$$\begin{aligned}\mathsf{Fill}^{\Psi|\delta\hookrightarrow\gamma}\ (A : \Pi z : \Psi.[\![\gamma\ z]\!]_{\mathsf{cof}} \to \mathsf{U}) := \quad & \Pi\alpha : \mathsf{Cof}. \\ & \Pi b : \Pi z : \Psi.[\![\delta\ z]\!]_{\mathsf{cof}} \to A\ z. \\ & \Pi t : \Pi z : \Psi.[\![\gamma\ z \wedge \alpha]\!]_{\mathsf{cof}} \to A\ z[\delta\ z \mapsto b]. \\ & \Pi z : \Psi.[\![\gamma\ z]\!]_{\mathsf{cof}} \to A\ z[\delta\ z \mapsto b, \alpha \mapsto t]\end{aligned}$$

The final technique we notice that is used to define some filling operations is to define them in the topos sliced over ([a subobject of] a product of) representables. To exemplify what this looks like, consider the filling predicate used to define the fibrant types in Cartesian cubical type theory.

$$
\begin{aligned}
\mathsf{comFill}^{\mathbb{I}}_{\mathrm{Cart}}\ (A : \mathbb{I} \to \mathsf{U}) := \quad & \Pi\alpha : \mathsf{Cof}. \\
& \Pi i : \mathbb{I}. \\
& \Pi b : A\ i. \\
& \Pi t : \Pi z : \mathbb{I}.[\![\alpha]\!]_{\mathsf{cof}} \to A\ z[z = i \mapsto b]. \\
& \Pi z : \mathbb{I}.A[z = i \mapsto b, \alpha \mapsto t] \\
\mathsf{comFill}_{\mathrm{Cart}}\ \Gamma\ (A : \Gamma \to \mathsf{U}) := \quad & \Pi i : \mathbb{I}.\Pi p : \mathbb{I} \to \Gamma.\mathsf{comFill}^{\mathbb{I}}_{\mathrm{Cart}}\ i\ (A \circ p)
\end{aligned}
$$

In this case, the trivial cofibration (or shape) of the filling operations consists of filling the interval from an *arbitrary* point along it, $i : \mathbb{I}$. For this to be possible, we think of the filling operation as being defined on the presheaf category sliced over the object $\mathbb{I}$, and not directly on the presheaf category itself. An important detail to note above is that the cofibration formula describing the generating trivial cofibration, $z = i$, is scoped by both the interval we slice over, $i : \mathbb{I}$, and the interval we are filling, $z : \mathbb{I}$.

At this point, we can abstract the above scenario and combine it with our previous abstractions to define our fully generalized notion of filling operation. As is the pattern in this chapter, instead of being sliced over the interval $\mathbb{I}$, we consider the presheaf category being sliced over an arbitrary product of representables in our presheaf Φ, and allow Φ to potentially be restricted by some cofibration $\Phi \vdash \varphi\ \mathsf{cofib}$. To update our previous abstractions to account for this change, we now consider filling the shape Ψ restricted by the cofibration formula $\Psi \vdash \gamma\ \mathsf{cofib}$ with the trivial cofibration described by the formula $\Phi, \Psi \vdash \delta\ \mathsf{cofib}$ where $\varphi \wedge \delta$ entails γ. All together, we get the following type for our filling operation.

As a final technicality, we specify a definition of a universe object. A formal definition is ultimately given in Definition 5.2.2, but to understand the following definition of a filling operation the only aspects of a universe object needed are to know that it is an object U that classifies other objects in the category and is equipped with a decode morphism El that converts terms in U (i.e. morphisms with codomain U) into the corresponding object of the category. We write $\mathcal{U}$ to denote the object classifying all objects in the category.

Definition 5.1.1. A *filling operation* for types in universe U_{F} with decode morphism El_{F} in the presheaf topos $\widehat{\mathbb{C}}$ is classified by a predicate $\mathsf{Fill}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi}$ with type $\Pi\Gamma : \mathcal{U}.(\Gamma \to \mathsf{U}_{\mathsf{F}}) \to \mathcal{U}$ indexed by the following:

- two objects that are both products of representables: Φ and Ψ;
- a cofibration formula scoped by Φ: $\Phi \vdash \varphi\,\mathsf{cofib}$;
- a cofibration formula scoped by Ψ: $\Psi \vdash \gamma\,\mathsf{cofib}$;
- a cofibration formula scoped by $\Phi \times \Psi$: $\Phi, \Psi \vdash \delta\,\mathsf{cofib}$.

Furthermore, the conjunction of φ and δ must entail γ: $\varphi \wedge \delta \vdash_{\Phi,\Psi} \gamma$. Using this data, we first define the type of *composition operations*.

$$
\begin{array}{lcl}
\mathsf{Comp}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi} & : & \Pi A : (\Pi z : \Psi.[\![\gamma\ z]\!]_{\mathsf{cof}} \to \mathsf{U_F}).\mathcal{U} \\
\mathsf{Comp}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi}\ A & := & \Pi\alpha : \mathsf{Cof}. \\
& & \Pi x : \Phi.\Pi * : [\![\varphi\ x]\!]_{\mathsf{cof}}. \\
& & \Pi b : \Pi z : \Psi.[\![\delta\ x\ z]\!]_{\mathsf{cof}} \to \mathsf{El_F}(A\ z\ *). \\
& & \Pi t : \Pi z : \Psi.[\![\gamma\ z \wedge \alpha]\!]_{\mathsf{cof}} \to \mathsf{El_F}(A\ z\ *)[\delta\ x\ z \mapsto b]. \\
& & \Pi z : \Psi.[\![\gamma\ z]\!]_{\mathsf{cof}} \to \mathsf{El_F}(A\ z\ *)[\delta\ x\ z \mapsto b, \alpha \mapsto t]
\end{array}
$$

We then define the predicate $\mathsf{Fill}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi}$ as shown below.

$$
\begin{array}{lcl}
\mathsf{Fill}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi} & : & \Pi\Gamma : \mathcal{U}.(\Gamma \to \mathsf{U_F}) \to \mathcal{U} \\
\mathsf{Fill}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi}\ \Gamma\ A & := & \Pi p : (\Pi z : \Psi.[\![\gamma\ z]\!]_{\mathsf{cof}} \to \Gamma). \\
& & \mathsf{Comp}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi}\ (A \circ p)
\end{array}
$$

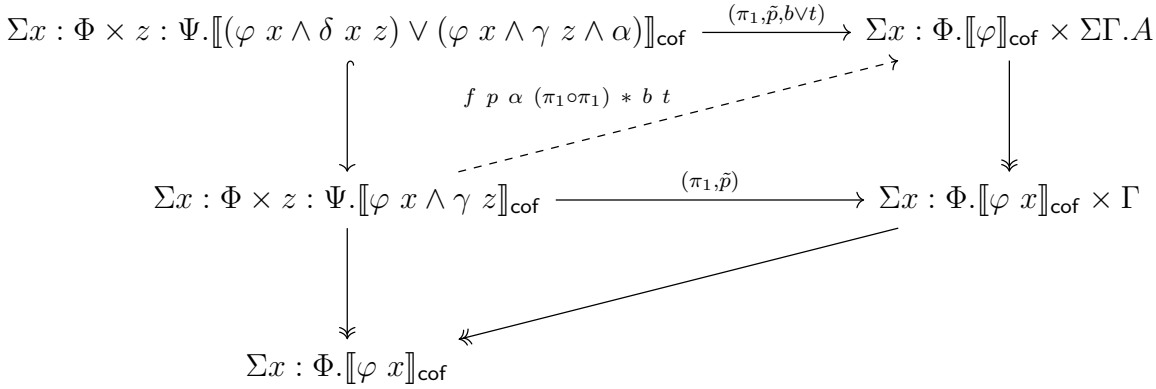
A filling operation with type $\mathsf{Fill}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi}\ \Gamma\ A$ conceptually corresponds to filling a type A along any inclusion of $\Sigma\Psi.[\![\delta]\!]_{\mathsf{cof}}$ into $\Sigma\Psi.[\![\gamma]\!]_{\mathsf{cof}}$ over any tope $\Sigma\Phi.[\![\varphi]\!]_{\mathsf{cof}}$ in context Γ. Note the fact that $\varphi \wedge \delta$ implies γ is required to ensure that the type of the base term b is well-formed.

Filling Operation	Φ	Ψ	φ	γ	δ
CCHM Kan	$\cdot$	$i : \mathbb{I}$	$\top$	$\top$	$i = \mathbb{0}_{\mathbb{I}}$
ABCFHL Kan	$i : \mathbb{I}$	$j : \mathbb{I}$	$\top$	$\top$	$i = j$
Bicubical Covariant	$\cdot$	$i : \mathbb{2}$	$\top$	$\top$	$i = \mathbb{0}_{\mathbb{2}}$
Bicubical Inner	$\cdot$	$i, j : \mathbb{2}$	$\top$	$i \leq j$	$i = \mathbb{0}_{\mathbb{2}} \vee j = \mathbb{1}_{\mathbb{2}}$
Contractible Fill*	$\cdot$	$\cdot$	$\top$	$\top$	$\bot$

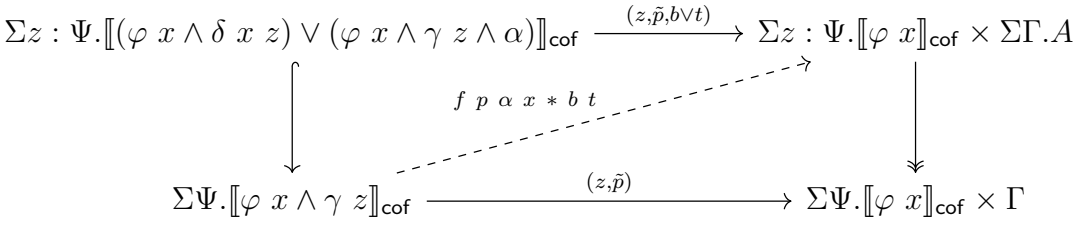
*The definition of contractible fill is isomorphic to the composition operation for the above data, and not the filling operation.

Figure 5.1: Instantiating the types of common filling operations using the generalized definition

To better understand the picture, let us first consider the what a solution to the filling operation looks like when we abstract (and thus slice) over the new tope $\Sigma\Phi.[\![\varphi]\!]_{\mathsf{cof}}$. As such, we consider the solution f as a morphism in the slice category $\widehat{\mathbb{C}}/(\Sigma\Phi.[\![\varphi]\!]_{\mathsf{cof}})$ for any given p, α, b and t. Do note that, in the below diagrams, $\tilde{p}$ is the morphism given by uncurrying p and precomposing p with the proper projections that correct the cofibration witness given by the domain of the morphism to that required by the diagram.



With the above diagram in mind, we can derive the representation to the filling problem precisely described by $f : \mathsf{Fill}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi}\ \Gamma\ A$ by substituting in a given morphism $\Gamma \vdash x : \Phi$ satisfying the cofibration φ. In doing so, we relocate the commutative square from the slice category to our original setting of the presheaf category $\widehat{\mathbb{C}}$.



A table summarizing how a number of common filling operations are encoded using Definition 5.1.1 is provided in Figure 5.1. While the cofibration φ is never used in these

common examples, we choose to include it in our definition as we think of the notion of a shape as being a representable paired with a cofibration, or what is often called a tope.

5.1.2 Contextual Filling and its Closure Conditions

In this section, we transition to exploring our final generalization of filling operations: They need not be able to fill over the entire context, and instead can depend on some of the context in a fiberwise fashion. To account for this, we introduce a modified version of the filling operation predicate that allows for the type family A to be scoped by two contexts: a "normal" context Γ and a fiberwise context on which A cannot be filled over, Ξ.

Definition 5.1.2. Given Φ, Ψ, φ, γ and δ as described in Definition 5.1.1, a *contextual filling operation* for types in $\mathsf{U_F}$ is classified by the predicate $\mathsf{FillCtx}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi}$ defined below.

$$\begin{aligned}
\mathsf{FillCtx}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi} \quad &: \quad \Pi\Xi : \mathsf{U}. \\
&\qquad \Pi\Gamma : \Xi \to \mathsf{U}. \\
&\qquad (\Pi x : \Xi.\ \Gamma\ x \to \mathsf{U_F}) \to \mathsf{U} \\
\mathsf{FillCtx}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi}\ \Xi\ \Gamma\ A \quad &:= \quad \Pi x : \Xi.\mathsf{Fill}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi}\ (\Gamma\ x)\ (A\ x)
\end{aligned}$$

Having defined what it means to be a contextual filling operation, we can now show some useful closure conditions that are always satisfied. For the remainder of this section, let us fix a filling problem given by $\mathsf{FillCtx}^{\Psi|\delta\hookrightarrow\gamma}_{\Phi|\varphi}$; for readability, we shall drop the parameters and simply denote the filling problem as $\mathsf{fillCtx}$. We also fix an arbitrary object Ξ and use the notation $*$ to denote the terms witnessing that a cofibration proposition is true.

Lemma 5.1.3. *Given an object Γ fibered over Ξ, an object A fibered over Ξ, and an object B fibered over Ξ, Γ and A if we have a morphism witnessing* $\mathsf{FillCtx}\ \Xi\ (A \times \Gamma)\ B$ *then we can construct a solution to* $\mathsf{FillCtx}\ (\Xi \times A)\ \Gamma\ B$.

Proof. This lemma is quite straightforward as it amounts to forgetting that one can fill over A. Thus, assume we have a term/morphism $\mathsf{fillB} : \mathsf{FillCtx}\ \Xi\ (A \times \Gamma)\ B$. We can construct the solution to $\mathsf{FillCtx}\ (\Xi \times A)\ \Gamma\ B$ using fillB by providing it with a constant path in A as shown here.

$$\lambda\ (x, a)\ p.\mathsf{fillB}\ x\ (\lambda z\ *.(p\ z\ *, a))$$

$\boxtimes$

Lemma 5.1.4. *Given an object A fibered over Ξ, an object Γ fibered over $\Xi \times A$, and an object B fibered over Ξ, Γ and A, if we have a morphism witnessing* $\mathsf{FillCtx}\ (\Xi \times A)\ \Gamma\ B$*, we can construct a solution to* $\mathsf{FillCtx}\ \Xi\ \Gamma\ \Pi x{:}A.B$.

Proof. From our hypothesis, we have a term/morphism $\mathsf{fillB} : \mathsf{FillCtx}\ (\Xi \times A)\ \Gamma\ B$. The desired term witnessing $\mathsf{FillCtx}\ \Xi\ \Gamma\ \Pi x{:}A.B$ is constructed as shown below.

$$\lambda\ x\ p\ \alpha\ y\ *\ b\ t\ z\ *\ a.\mathsf{fillB}\ x\ p\ \alpha\ y\ *\ (\lambda z\ *.b\ z\ *\ a)\ (\lambda z\ *.t\ z\ *\ a)\ z\ *$$

In other words, we can use the solution fillB directly, simply instantiating the provided b and t of the Π-type at the argument a. $\boxtimes$

Corollary 5.1.5. *Given an object Γ fibered over Ξ, and an object A fibered over Ξ, Γ and the subobject classifier Ω, if we have a morphism witnessing* $\mathsf{FillCtx}\ (\Xi \times \Omega)\ \Gamma\ A$*, we can construct a solution to* $\mathsf{FillCtx}\ \Xi\ \Gamma\ \Pi x{:}\Omega.A$.

Corollary 5.1.6. *Given a subobject formula ξ of Ξ, an object Γ fibered over Ξ, and an object A fibered over Ξ, Γ and the proof witness $[\xi]$, if we have a morphism satisfying* $\mathsf{FillCtx}\ (\Xi \times [\xi])\ \Gamma\ A$*, we can construct a solution to* $\mathsf{FillCtx}\ \Xi\ \Gamma\ ([\xi] \to A)$.

Lemma 5.1.7. *Given an object Γ fibered over Ξ, an object A fibered over Ξ and Γ, and an object B fibered over Ξ, Γ and A if we have a morphism witnessing* $\mathsf{FillCtx}\ \Xi\ \Gamma\ A$ *and a morphism witnessing* $\mathsf{FillCtx}\ \Xi\ (\Gamma \times A)\ B$ *we can construct a solution to* $\mathsf{FillCtx}\ \Xi\ \Gamma\ \Sigma x : A.B$.

Proof. Let fillA be the witness to $\mathsf{FillCtx}\ \Xi\ \Gamma\ A$ and fillB that of $\mathsf{FillCtx}\ \Xi\ (\Gamma \times A)\ B$. The solution for the Σ-type amounts to pairing together the solutions provided by fillA and fillB. We can define the solution as follows, using let notation to simplify the definition, and simply interpreting it as syntactic sugar for the term that results from inlining the let-bound variable

$$\lambda\ x\ p\ \alpha\ y\ *\ b\ t\ z\ *.\ \ \mathsf{let}\ a = \mathsf{fillA}\ x\ p\ \alpha\ y\ *\ (b \circ \mathsf{fst})\ (t \circ \mathsf{fst})\ \mathsf{in}$$
$$(a\ z\ *, \mathsf{fillB}\ x\ (\lambda z\ *.(p\ z\ *, a\ z\ *))\ \alpha\ y\ *\ (b \circ \mathsf{snd})\ (t \circ \mathsf{snd}))$$

⊠

Lemma 5.1.8. *Assume we have a subobject formula ξ of Ξ, an object Γ fibered over Ξ, an object A fibered over Ξ and Γ, and a section a of A over $\Sigma\Xi.\Sigma[\xi].\Gamma$. If we have a morphism satisfying* $\mathsf{FillCtx}\ \Xi\ \Gamma\ A$*, we can construct a solution to* $\mathsf{FillCtx}\ \Xi\ \Gamma\ A[\xi \mapsto a]$.

Proof. In this case, it ends up that the witness induced by $\mathsf{fillA} : \mathsf{FillCtx}\ \Xi\ \Gamma\ A$ provides the solution for $\mathsf{FillCtx}\ \Xi\ \Gamma\ A[\xi \mapsto a]$. The only aspect we must pay particular attention to is that the boundary condition is constantly satisfied. At a high level, the key observation is that, while the type A may vary along the shape we are filling, as the cofibration ξ used in the boundary constraint only depends on Ξ, that "portion" of A remains constant throughout the filling process. Taking advantage of this fact, we can construct the solution as shown here.

$$\lambda\ x\ p\ \alpha\ y\ *\ b\ t.\mathsf{fillA}\ x\ p\ (\alpha \vee \xi\ x)\ y\ *\ b\ (t \vee a)$$

The b and t provided to the function above already are equal to a along ξ by definition, and thus $t \vee a$ is a well-formed elimination from the disjunction $\alpha \vee \xi\ x$, and is also compatible with b along the cofibration δ. Furthermore, by providing $\alpha \vee \xi\ x$ as the cofibration to the filling operation, the output must also satisfy the boundary condition. The only reason we can provide ξ as an argument to the solution is that it only depends on Ξ, which is in scope via the argument x; were ξ to depend on Γ as well, it would not be possible to express ξ using what is available in such a way that the solution is well formed (one could attempt to do so by further unwrapping the solution function and providing the arguments of type Ψ and $[\![\gamma]\!]_{\mathsf{cof}}$ to the path p, but the resulting term of type Γ would not match up with that used in the instance of ξ present in the base b and tube t, as in b and t the corresponding arguments are quantified by a Π and thus are distinct from those given as arguments to the filling operation). ⊠

5.2 A Type Theory with Internally Definable Universes

An exciting application of this work (that in particular motivated this foray into contextual extension types) is that it greatly aids the creation of a reasonable syntax of internally definable universes given by the LOPS construction [47], exposing the ability to define new classifying universes to the syntax of the type theory. The general idea is as follows: As a user of a type theory, say you have some universe U, representable object Ψ and have a predicate $P : (\Psi \to \mathsf{U}) \to \mathsf{U}$ that picks out types with some property you care greatly about (e.g. U is the universe of Kan types, and $P : (2 \to \mathsf{U}) \to \mathsf{U}$ describes which Kan types are covariant fibrations), and you wish to define the universe of types equipped with a witness to property P. Instead of requiring the implementation and/or rules to directly describe this new universe U_P along with the corresponding filling operations for every type former that inhabits U_P, the idea we present in this section is that the type theory is equipped with a general universe constructor which takes the outer universe U and predicate P as input and automatically constructs the universe $\mathsf{Univ}(\mathsf{U}, P)$, the LOPS universe of types in U that additionally are equipped with witnesses to predicate P. The user can then internally provide filling operations for the types that inhabit U_P within the type theory itself, simplifying that which must be described in the rules of the theory and most notably greatly reducing the laborious task of defining filling algorithms as a part of the implementation of a proof assistant. The challenge with doing this internally is that the user provided predicate P must land in the universe U that is being restricted to define the new universe, and thus one must know that P can always be equipped with a solution to whatever filling problem(s) U classifies; thankfully, the results given in this section will ultimately allow us to do just that.

5.2.1 Judgement Forms

Again working in the internal logic of a presheaf topos indexed by a category $\mathbb{C}$, we use the following notation and judgments:

- Ξ and Γ are *contexts*, interpreted as (products of) objects in the topos.
- We use boldface variables $\mathbf{A}$ to indicate types corresponding to representable objects, and we occasionally write $\mathbf{A}\ \mathsf{Rep}$ do denote the judgement that $\mathbf{A}$ is such a type. We use the judgement $\Gamma \vdash a : \mathbf{A}$ to say a is a presheaf morphism from Γ to $\mathbf{A}$, and call a a *representable term*.
- $\Gamma \vdash \alpha\ \mathsf{cofib}$ is a *cofibration*, a special kind of logical formula describing subobjects of (products of) representables used in filling operations. We interpret α as a morphism from Γ into Cof.
- $\varphi \vdash_\Gamma \psi$ is the implication ordering of cofibrations on subobjects of Γ.
- $\Gamma \vdash \alpha$ is the judgement stating that the cofibration α is true assuming the context Γ.
- $\Gamma \vdash A\ \mathsf{Type}_i$ indicates A is a type in context Γ. We suppress levels i throughout, but formally types are stratified by size. We interpret the judgement to indicate that A is a presheaf in the slice topos $\widehat{\mathbb{C}}/\Gamma$.
- $\Gamma \vdash a : A$ is the judgement indicating a is a term of a type A relative to context Γ. This corresponds to a representing a morphism from Γ to A in the topos.
- Lastly, $\Gamma \vdash a_0 \equiv a_1 : A$ is the denotes that a_0 and a_1 are judgmentally equal terms of type A in context Γ. We may simply write $a_0 \equiv a_1$ as a shorthand when Γ and A can easily be inferred from context.

5.2.2 Typing Rules

We will assume the standard typing rules that hold in the internal logic of every topos, in particular for Π- and Σ-types. In addition, we will focus on the following rules that pertain to representable objects and cofibrations.

Representable Types

First, we will specify syntax isolating the representable objects as types, using a bold font to distinguish them visually. As we are working at such a high level of generality here (i.e. in an arbitrary presheaf topos $\widehat{\mathbb{C}}$), we simply will assume the existence of a syntax for terms of representables corresponding to the morphisms contained within the topos.

$$\frac{\mathbf{A} \in \mathbb{C}}{\mathbf{A}\ \mathsf{Rep}} \qquad \frac{\mathbf{A}\ \mathsf{Rep}}{\cdot \vdash \mathbf{A}\ \mathsf{Type}}$$

$$\frac{a \in \widehat{\mathbb{C}}(\Gamma, y\mathbf{A})}{\Gamma \vdash a : \mathbf{A}}$$

Cofibration Types

We now fix a syntax for working with cofibrations in the type theory. To do so, we specify a type Cof corresponding to the cofibration classifier in the topos, and thus the terms of Cof are cofibration formulae.

$$\frac{}{\cdot \vdash \mathsf{Cof}\ \mathsf{Type}} \qquad \frac{\Gamma \vdash \alpha\ \mathsf{cofib}}{\Gamma \vdash \alpha : \mathsf{Cof}}$$

Given a cofibration formula α, we also introduce the type $[\alpha]$ that witnesses the when α is satisfied, semantically corresponding to the right projection of the subobject pullback of α. As the type $[\alpha]$ is a strict proposition, we simply inhabit it with a single term $*$ when the formula is true. In practice, implementations often make all cofibration proof terms implicit in the syntax, with the type checker algorithm simply verifying the formula holds true when a term of one is required.

$$\frac{\Gamma \vdash \alpha\ \mathsf{cofib}}{\Gamma \vdash [\alpha]\ \mathsf{Type}} \qquad \frac{\Gamma \vdash \alpha}{\Gamma \vdash * : [\alpha]}$$

Boundary Types

The last relatively standard type former we wish to hone in on are boundary types, also called extension types in the literature. A boundary type pairs together a type A, a cofibration α, and a partial term a in A assuming α and results in a type containing the total terms of A that are strictly equal to the partial term a when the cofibration α is true.

$$\frac{\Gamma \vdash A\ \mathsf{Type} \quad \Gamma \vdash \alpha\ \mathsf{cofib} \quad \Gamma, x : [\alpha] \vdash a : A}{\Gamma \vdash A[\alpha \mapsto x.a]\ \mathsf{Type}}$$

$$\frac{\Gamma \vdash a' : A \quad \Gamma, x : [\alpha] \vdash a \equiv a' : A}{\Gamma \vdash a' : A[\alpha \mapsto x.a]}$$

5.2.3 Syntax for Internal Universes

For this notion of internal LOPS universes, we parameterize the universe by components corresponding to that which specifies a filling operation as defined in Definition 5.1.1:

1. A universe U from which our new universe will be carved;
2. A representable $\mathbf{A}$ and cofibration φ scoped by A over which the filling operation is sliced;
3. A representable $\mathbf{B}$, a cofibration γ scoped by B and a cofibration δ scoped by A and B where δ entails γ assuming φ (as an important note, we will ultimately be refining how we define γ in the formal rules we propose);
4. A witness that the composition operation specified by the above data is contained in the universe U.

Given these parameters, the idea is that the universe specified by the above parameters represents the subuniverse of U containing those types for which we have a filling operation that can always fill the shape $\mathbf{B}$ restricted by γ from inputs of shape $\mathbf{B}$ restricted by δ, all sliced over the shape $\mathbf{A}$ restricted by φ.

The hypotheses assumed by by the LOPS construction force us to maintain a few requirements in our proposal in order for this syntax to accurately and safely match the construction:

1. The category $\widehat{\mathbb{C}}$ must be closed under finite products;

2. The object $\Sigma\mathbf{B}.[\gamma]$ must be tiny.

Do note that, while these requirements are shown sufficient by the LOPS construction in order to define a universe constructively, they need not be required, and one may choose to use a version of the proposed syntax but loosen these requirements and justify the existence of the universe using alternative means (e.g. should one wish to incorporate user defined universes in a type theory for which one does not care about constructivity).

Putting this all together results in incorporating the syntax and inference rules shown in Figure 5.2 to the type theory.

Before diving into the rules themselves, we need to first explain the nonstandard judgements: $\mathsf{U}\ \mathsf{Univ}$, $\mathbf{A}\ \mathsf{Rep}, \mathsf{FillData}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)$ and $\mathsf{UData}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)$. The first judgement conveys that U is a universe of types. The idea is that, first, the outermost universe $\mathcal{U}$ is a universe, and every new universe defined by these rules is also a universe. In addition, there is a function El_{U} that sends elements of the universe U to their underlying type. We can express this formally as shown here.

$$\frac{}{\mathcal{U}\ \mathsf{Univ}}$$

$$\frac{}{\mathsf{Univ}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \mathsf{Univ}}$$

$$\frac{\mathsf{U}\ \mathsf{Univ} \qquad \Gamma \vdash A : \mathsf{U}}{\Gamma \vdash \mathsf{El}_{\mathsf{U}}\ A\ \mathsf{Type}}$$

$$\frac{\Gamma \vdash A : \mathcal{U}}{\mathsf{El}_{\mathcal{U}}\ A \equiv A}$$

$$\frac{\Gamma \vdash \mathsf{code}(\mathsf{U}, \ldots)(\Delta, A, \mathsf{fillA})\ x : \mathsf{Univ}(\mathsf{U}, \ldots)}{\mathsf{El}_{\mathsf{Univ}(\mathsf{U},\ldots)}\ (\mathsf{code}(\mathsf{U}, \ldots)(\Delta, A, \mathsf{fillA})\ x) \equiv \mathsf{El}_{\mathsf{U}}\ (A\ x)}$$

$$\frac{\mathsf{UData}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)}{\vdash \mathsf{Univ}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \mathsf{Type}}$$

$$\frac{\mathsf{UData}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)}{\cdot \vdash \mathsf{El}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) : \mathsf{Univ}(\mathsf{U}, \ldots) \to \mathsf{U}}$$

$$\frac{\mathsf{UData}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)}{\begin{array}{c}\cdot \vdash \mathsf{fill}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) : \Pi p : (\Pi y : \mathbf{B}.[y = \beta\ y] \to \mathsf{Univ}(\mathsf{U}, \ldots)). \\ \mathsf{Fill}(\mathsf{U}, \ldots)\ (\mathsf{El}(\mathsf{U}, \ldots) \circ p)\end{array}}$$

$$\frac{\mathsf{UData}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)}{\cdot \vdash \mathsf{Fill}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) : \Pi A : (\Pi y : \mathbf{B}.[y = \beta\ y] \to \mathsf{U}).\mathsf{U}}$$

$$\frac{\begin{array}{c}\mathsf{UData}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) \\ \cdot \vdash \Gamma\ \mathsf{Type} \qquad \cdot \vdash A : \Gamma \to \mathsf{U} \\ \cdot \vdash \mathsf{fillA} : \Pi p : (\Pi y : \mathbf{B}.[y = \beta\ y] \to \Gamma).\mathsf{El}_{\mathsf{U}}\ (\mathsf{Fill}(\mathsf{U}, \ldots)\ (A \circ p))\end{array}}{\cdot \vdash \mathsf{code}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)(\Gamma, A, \mathsf{fillA}) : \Gamma \to \mathsf{Univ}(\mathsf{U}, \ldots)}$$

$$\mathsf{El}(\mathsf{U}, \ldots)\ (\mathsf{code}(\mathsf{U}, \ldots)(\Gamma, A, \mathsf{fillA})\ x)\ \equiv\ A\ x$$

$$\mathsf{fill}(\mathsf{U}, \ldots)\ (\lambda y\ *.\mathsf{code}(\mathsf{U}, \ldots)(\Gamma, A, \mathsf{fillA})\ (p\ y\ *))\ \equiv\ \mathsf{fillA}\ (p\ y\ *)$$

Figure 5.2: Rules for internal LOPS universes

The second judgement, $\mathbf{A}\ \mathsf{Rep}$, simply indicates that $\mathbf{A}$ is a type corresponding to a representable. It is worthwhile to note that, as here we are choosing to directly follow the LOPS construction, the category $\mathbb{C}$ is closed with respect to finite products and thus obviously any product of representables is itself a representable; should one wish to modify this approach for a setting in which this isn't the case, one might consider modifying this judgement to instead capture the situation where $\mathbf{A}$ corresponds to a product of representables in the topos.

We also introduce judgements $\mathsf{FillData}$ and UData which simply are shorthand grouping together all of the hypotheses required of the components used to specify a new filling problem or universe. The judgements are defined with the following inference rules.

Filling Operation	**A**	**B**	φ	β	δ
CCHM Kan	$\cdot$	$i : \mathbb{I}$	$\top$	$\lambda x.x$	$i = \mathbb{0}_{\mathbb{I}}$
ABCFHL Kan	$i : \mathbb{I}$	$j : \mathbb{I}$	$\top$	$\lambda x.x$	$i = j$
Bicubical Covariant	$\cdot$	$i : \mathbb{2}$	$\top$	$\lambda x.x$	$i = \mathbb{0}_{\mathbb{2}}$
Bicubical Inner	$\cdot$	$i, j : \mathbb{2}$	$\top$	$\lambda(x, y).(x \wedge y, y)$	$i = \mathbb{0}_{\mathbb{2}} \vee j = \mathbb{1}_{\mathbb{2}}$

Figure 5.3: Instantiating the types of common filling operations as filling data in the syntax

$$\frac{\begin{array}{c}\mathbf{A}\ \mathsf{Rep} \qquad \mathbf{B}\ \mathsf{Rep} \\ x : \mathbf{A} \vdash \varphi\ x\ \mathsf{cofib} \qquad y : \mathbf{B} \vdash \beta\ y : \mathbf{B} \qquad \beta\ (\beta\ y) \equiv \beta\ y \\ x : \mathbf{A}, y : \mathbf{B} \vdash \delta\ x\ y\ \mathsf{cofib} \qquad \varphi\ x \wedge \delta\ x\ y \vdash_{x:\mathbf{A},y:\mathbf{B}} y = \beta\ y\end{array}}{\mathsf{FillData}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)}$$

$$\frac{\begin{array}{rl} & \mathsf{U}\ \mathsf{Univ} \qquad \mathsf{FillData}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) \\ \cdot \vdash \Pi\mathsf{lem} : & \Pi\Xi : \mathcal{U}.\Pi\Gamma : \Xi \to \mathcal{U}. \\ & \Pi A : \Pi u : \Xi.\Gamma\ u \to \mathcal{U}. \\ & \Pi B : \Pi u : \Xi.\Pi v : \Gamma\ u.A\ u\ v \to \mathcal{U}. \\ & \mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi\ \Gamma\ A \to \\ & \mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi \\ & \qquad\qquad (\Sigma v : \Gamma\ u.A\ u\ v) \\ & \qquad\qquad (\lambda u\ (v, w).B\ u\ v\ w) \to \\ & \mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi\ \Gamma \\ & \qquad\qquad (\lambda u\ v.\Pi w : A\ u\ v.B\ u\ v\ w)\end{array}}{\mathsf{UData}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)}$$

We will derive an internally definable element of the universe U that will act as the definition of the type $\mathsf{Fill}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ x\ *\ A$ by the end of Section 5.2.4, but for our initial discussion of the rules, all that matters is knowing the underlying type it decodes to, shown here.

$$\begin{array}{rl} \mathsf{El}_{\mathsf{U}}\ (\mathsf{Fill}(\mathsf{U}, \ldots)\ A) := & \Pi\alpha : \mathsf{Cof}. \\ & \Pi x : \mathbf{A}.\Pi * : [\varphi\ x]. \\ & \Pi b : \Pi y : \mathbf{B}.[\delta\ x\ y] \to A\ y\ *. \\ & \Pi t : \Pi y : \mathbf{B}.[(y = \beta\ y) \wedge \alpha] \to A\ y\ *[\delta\ x\ y \mapsto b]. \\ & \Pi y : \mathbf{B}.[y = \beta\ y] \to A\ y\ *[\delta\ x\ y \mapsto b, \alpha \mapsto t] \end{array}$$

The rules themselves are relatively straightforward, as they correspond very closely to the LOPS construction for the notion of filling operations defined in Definition 5.1.1. For the

formation rule, we define a new universe, $\mathsf{Univ}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)$, which will contain those types in an existing universe U that can be equipped with a solution to $\mathsf{Fill}^{y:\mathbf{B}|\delta xy \hookrightarrow (y=\beta xy)}_{x:\mathbf{A}|\varphi x}$ where the hypotheses to the formation rule are the same as those to the definition of the filling operation. The one important detail to note is β: instead of directly providing the cofibration restricting $\mathbf{B}$ to give the shape being filled, the universe constructor instead requests an idempotent function from $\mathbf{B}$ to itself (sliced over $\mathbf{A}$ restricted by φ), and the cofibration that describes the shape is given by the equalizer of the identity function on $\mathbf{B}$ with this function (i.e. the image of β). As an object in a presheaf topos is tiny if and only if it is the image of an idempotent representable functor (i.e. the retract of a representable functor), the constructor asks for an idempotent representable functor directly to ensure the resulting object is tiny with the guarantee that every tiny object is describable in this way. The new universe comes with a decode function $\mathsf{El}(\mathsf{U}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)$. As expected, decoding a code gives the original type, enforced by the first equation in Figure 5.2. Similarly, encoding the decode of a type gives the original element of the universe, as indicated by the final equation in Figure 5.2. The decode function is also equipped with a solution to its corresponding filling problem, given by $\mathsf{fill}(\mathsf{U}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)$. For any *closed* type Γ and *closed* type family $A : \Gamma \to \mathsf{U}$, the type of the filling operation for A is given by $\mathsf{Fill}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Gamma\ A$, and is defined to be precisely the type corresponding to the definition of $\mathsf{Fill}^{y:\mathbf{B}|\delta xy \hookrightarrow (y=\beta xy)}_{x:\mathbf{A}|\varphi x}$. Lastly, we also provide an encode function for the universe: Given a *closed* type Γ, a *closed* type family $A : \Gamma \to \mathsf{U}$, and a filling operation for A denoted by fillA, the term $\mathsf{code}(\mathsf{U}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)(\Gamma, A, \mathsf{fillA})$ is a function from Γ into the new universe. The restriction to closed types for the encode function corresponds to the use of the flat modality in [47], and will be explained in more detail when we work out the semantics of this theory in Section 5.2.5.

Let us take a paragraph to focus on the filling operation for the universe, the rules pertaining to it, and how they work together to provide the behavior we anticipate for the inhabitants of our new universe. Note that these rules are pulled directly from the

LOPS construction, and thus are described in greater detail in [47]. The filling operation for the decode function actually grants access to the filling operations for all types in the new universe: Given a term $B : A \to \mathsf{Univ}(\mathsf{U}, \ldots)$, if we wish to fill B along a shape $p : \Pi y : \mathbf{B}.[y = \beta\ y] \to A$, the filling operation we need is given by $\mathsf{fill}(\mathsf{U}, \ldots)\ (B \circ p)$. Note that, if B is of the form $\mathsf{code}(\mathsf{U}, \ldots)(\Gamma, A, \mathsf{fillA}) \circ f$, the filler $\mathsf{fill}(\mathsf{U}, \ldots)\ (B \circ p)$ reduces to $\mathsf{fillA}\ (f \circ p)$ by the second equation in Figure 5.2, and thus filling in a code in the universe does indeed reduce to using the filler provided with the code.

5.2.4 Encoding the Fill Type and Fiberwise Reasoning

As mentioned earlier, $\mathsf{Fill}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ A$ is to be defined as an internally expressible element of the universe $\mathcal{U}$; in order to do so, we must internalize the notion of fiberwise, contextual filling introduced in Section 5.1.2. Having done so, we can then build up the required filling operations (and thus codes) for Fill out of modular constructions defined over contextual filling.

$$\frac{\begin{array}{c}\mathsf{FillData}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) \\ \Delta \vdash \Xi\ \mathsf{Type} \qquad \Delta \vdash \Gamma : \Xi \to \mathcal{U} \qquad \Delta \vdash A : \Pi u : \Xi.\Gamma\ u \to \mathcal{U}\end{array}}{\Delta \vdash \mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi\ \Gamma\ A\ \mathsf{Type}}$$

$$\begin{array}{l}\mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi\ \Gamma\ A \equiv \\ \quad \Pi u : \Xi. \\ \quad \Pi p : (\Pi y : \mathbf{B}.[y = \beta\ y] \to \Gamma\ u). \\ \quad \Pi \alpha : \mathsf{Cof}. \\ \quad \Pi x : \mathbf{A}.\Pi * : [\varphi\ x]. \\ \quad \Pi b : \Pi y : \mathbf{B}.[\delta\ x\ y] \to A\ u\ (p\ y\ *). \\ \quad \Pi t : \Pi y : \mathbf{B}.[(y = \beta\ y) \wedge \alpha] \to A\ u\ (p\ y\ *)[\delta\ x\ y \mapsto b]. \\ \quad \Pi y : \mathbf{B}.[y = \beta\ y] \to A\ u\ (p\ y\ *)[\delta\ x\ y \mapsto b, \alpha \mapsto t]\end{array}$$

Figure 5.4: Typing rules for fiberwise filling operations

To begin this process, we introduce a type constructor for contextual filling operations and pick out the type to which it is equal in Figure 5.4. As this type will be used to aid in the definition of terms of the Fill type, we do not need it to land in any particular universe

and thus only consider it as an element of $\mathcal{U}$; in doing so, we have no issues immediately defining the internal type it represents. It also will become useful to note that ultimately the following equation holds (assuming a closed context Δ).

$$\begin{aligned}\mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi\ \Gamma\ A \equiv\ & \Pi u : \Xi. \\ & \Pi p : (\Pi y : \mathbf{B}.[y = \beta\ y] \to \Gamma\ u). \\ & \mathsf{El_U}\ (\mathsf{Fill}(\mathsf{U}, \ldots)\ ((A\ u) \circ p))\end{aligned}$$

Taking this equation into account, one can see the definition is nearly identical to that given in Section 5.1.2. This also demonstrates that when Ξ is the empty context/unit type, FillCtx and the type of the filling argument required by the code constructor are isomorphic (distinguished by a trivial quantification over the unit type).

The next step is to write down the rules we can stitch together to define filling operations for types that depend on fiberwise contextual reasoning (such as the Fill type). We do so in Figure 5.5. As with FillCtx, these rules are internally derivable and thus we define them to be equal to terms already present in the theory in Figure 5.6. Akin to the definition of FillCtx, these rules follow the first section of this chapter very closely, in particular corresponding to the lemmas in Section 5.1.2; in this case, they match to such a great extent that the same proofs justify the equations are correct and the terms type check soundly. The one case that doesn't precisely match up to anything cfillU, but thankfully its justification is straightforward: If a type A is already in the universe, it contains a solution to the filling problem for its entire context. Also, as we know from Lemma 5.1.3, we can weaken the fibrant context and forget how to fill in portions of the context. Looking at the definition of the term cfillU in Figure 5.6, we see it combines these two facts to conveniently take any type that is fully fibrant and provide a witness to the fact it is contextually fibrant for the desired contexts.

As a final step, we will make these lemmas easier to work with by defining wrappers containing both the type they construct along with the corresponding proof of fiberwise fibrancy using a Σ-type. We define the type of the wrappers cFib as follows.

$$\frac{\begin{array}{c}\mathsf{FillData}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) \\ \Delta \vdash \Xi\ \mathsf{Type} \qquad \Delta \vdash \Gamma : \Xi \to \mathcal{U} \qquad \Delta \vdash A : \Xi \to \mathcal{U} \\ \Delta \vdash B : \Pi u : \Xi . A\ u \to \Gamma\ u \to \mathcal{U} \\ \Delta \vdash \mathsf{fillB} : \mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ (\Sigma u : \Xi . A\ u) \\ \qquad\qquad (\Gamma \circ \mathsf{fst}) \\ \qquad\qquad (\lambda(u, v).B\ u\ v)\end{array}}{\begin{array}{r}\Delta \vdash \mathsf{cfillΠ}(\Xi, \Gamma, A, B, \mathsf{fillB}) : \mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi\ \Gamma \\ (\lambda u\ v . \Pi w : A\ u . B\ u\ w\ v)\end{array}}$$

$$\frac{\begin{array}{c}\mathsf{FillData}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) \\ \Delta \vdash \Xi\ \mathsf{Type} \qquad \Delta \vdash \Gamma : \Xi \to \mathcal{U} \qquad \Delta \vdash A : \Pi u : \Xi . \Gamma\ u \to \mathcal{U} \\ \Delta, u : \Xi \vdash \alpha\ u\ \mathsf{cofib} \qquad \Delta, u : \Xi, v : \Gamma, * : [\alpha] \vdash a\ u\ v : A \\ \Delta \vdash \mathsf{fillA} : \mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi\ \Gamma\ A\end{array}}{\begin{array}{r}\Delta \vdash \mathsf{cfillBdry}(\Xi, \Gamma, A, \alpha, a, \mathsf{fillA}) : \mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi\ \Gamma \\ (\lambda u\ v . A\ u\ v[\alpha\ u \mapsto a\ u\ v])\end{array}}$$

$$\frac{\begin{array}{c}\mathsf{FillData}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) \\ \Delta \vdash \Xi\ \mathsf{Type} \qquad \Delta \vdash \Gamma : \Xi \to \mathcal{U} \qquad \Delta \vdash A : \Pi u : \Xi . \Gamma\ u \to \mathcal{U} \\ \Delta \vdash B : \Pi u : \Xi . \Pi v : \Gamma\ u . A\ u\ v \to \mathcal{U} \\ \Delta \vdash \mathsf{fillA} : \mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi\ \Gamma\ A \\ \Delta \vdash \mathsf{fillB} : \mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi \\ \qquad\qquad (\Sigma v : \Gamma\ u . A\ u\ v) \\ \qquad\qquad (\lambda u\ (v, w).B\ u\ v\ w)\end{array}}{\begin{array}{r}\Delta \vdash \mathsf{cfillΣ}(\Xi, \Gamma, A, B, \mathsf{fillA}, \mathsf{fillB}) : \mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi\ \Gamma \\ (\lambda u\ v . \Sigma w : A\ u\ v . B\ u\ v\ w)\end{array}}$$

$$\frac{\begin{array}{c}\mathsf{FillData}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) \\ \Delta \vdash \Xi\ \mathsf{Type} \qquad \Delta \vdash \Gamma : \Xi \to \mathcal{U} \qquad \Delta \vdash A : \Pi u : \Xi . \Gamma\ u \to \mathsf{Univ}(\mathsf{U}, \ldots)\end{array}}{\Delta \vdash \mathsf{cfillU}(\Xi, \Gamma, A) : \mathsf{FillCtx}(x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi\ \Gamma\ (\mathsf{El}_{\mathsf{Univ}(\ldots)} \circ A)}$$

Figure 5.5: Closure conditions for fiberwise filling operations

$$\mathsf{cfillΠ}(\Xi, \Gamma, A, B, \mathsf{fillB}) \equiv$$
$$\lambda u\ p\ \alpha\ x\ *\ b\ t\ y\ *\ a.\mathsf{fillB}\ u\ x\ *\ p\ \alpha\ (\lambda y\ *.b\ x\ *\ a)\ (\lambda y\ *.t\ x\ *\ a)\ y\ *$$

$$\mathsf{cfillBdry}(\Xi, \Gamma, A, \psi, a, \mathsf{fillA}) \equiv$$
$$\lambda u\ p\ \alpha\ x\ *\ b\ t.\mathsf{fillA}\ u\ p\ (\alpha \vee \psi)\ x\ *\ b\ (t \vee a)$$

$$\mathsf{cfillΣ}(\Xi, \Gamma, A, B, \mathsf{fillB}) \equiv$$
$$\lambda u\ p\ \alpha\ x\ *\ b\ t\ y\ *.\ \mathsf{let}\ a = \mathsf{fillA}\ u\ p\ \alpha\ x\ *\ (b \circ \mathsf{fst})\ (t \circ \mathsf{fst})\ \mathsf{in}$$
$$(a\ y\ *, \mathsf{fillB}\ u\ (\lambda y\ *.(p\ y\ *, a\ y\ *))\ \alpha\ x\ *\ (b \circ \mathsf{snd})\ (t \circ \mathsf{snd}))$$

$$\mathsf{cfillU}(\Xi, \Gamma, A) \equiv$$
$$\lambda u\ p.\mathsf{fill}(\ldots)\ (\Sigma u : \Xi.\Gamma\ u)\ (\lambda(u, v).A\ u\ v)\ (\lambda y\ *.(u, p\ y\ *))$$

Figure 5.6: Equations defining fiberwise filling closure condition terms

$$\mathsf{cFib}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) \quad : \quad \Pi\Xi : \mathcal{U}.(\Xi \to \mathcal{U}) \to \mathcal{U}$$
$$\mathsf{cFib}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi\ \Gamma :=$$
$$\Sigma A : (\Pi u : \Xi.\Gamma\ u \to \mathcal{U}).\mathsf{FillCtx}(\mathsf{U}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ \Xi\ \Gamma\ A$$

Using this type, we can construct the definitions in Figure 5.7 that summarize the contents of Figures 5.5 and 5.6. To ultimately further simplify the notation, we use brackets to indicate implicit arguments.

At this point, we can finally construct the term interpreting the original formation rule for $\mathsf{Fill}(\ldots)$ given in Figure 5.2.

$$\cdot \vdash \mathsf{Fill}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) : \Pi A : (\Pi y : \mathbf{B}.[y = \beta\ y] \to \mathsf{U}).\mathsf{U}$$

First, we define a cFib corresponding to the filling operations type.

$$\cdot \vdash \mathsf{cFill}(\mathsf{U}', \Pi\mathsf{lem}', x : \mathbf{A}', y : \mathbf{B}', \varphi'\ x, \beta'\ y, \delta'\ x\ y) :$$
$$\mathsf{cFib}(\mathsf{U}', \Pi\mathsf{lem}', x : \mathbf{A}', y : \mathbf{B}', \varphi'\ x, \beta'\ y, \delta'\ x\ y)\ 1\ (\lambda\_.\Pi y : \mathbf{B}.[y = \beta\ y] \to \mathsf{U})$$

We define $\mathsf{cFill}(\ldots)$ as the following term.

$$
\begin{array}{lcl}
\mathsf{ctx}\Pi & : & \Pi\{\Xi\} : \mathcal{U}. \\
 & & \Pi\{\Gamma\} : \Xi \to \mathcal{U}. \\
 & & \Pi A : \Xi \to \mathcal{U}. \\
 & & \mathsf{cFib}(\ldots)\ (\Sigma u : \Xi.A\ u)\ \Gamma \to \mathsf{cFib}(\ldots)\ \Xi\ \Gamma \\
\mathsf{ctx}\Pi\ \{\Xi\}\ \{\Gamma\}\ A\ (B, \mathsf{fillB}) & := & (\lambda u\ v.\Pi w : A\ u.B\ (u, w)\ v, \\
 & & \quad \mathsf{cfill}\Pi(\Xi, \Gamma, A, B, \mathsf{fillB})) \\
 & & \\
\mathsf{ctx}\Pi_2 & : & \Pi\{\Xi\} : \mathcal{U}. \\
 & & \Pi\{\Gamma\} : \Xi \to \mathcal{U}. \\
 & & \Pi(A, \mathsf{fillA}) : \mathsf{cFib}(\mathsf{U}, \Pi\mathsf{lem}, \ldots)\ \Xi\ \Gamma. \\
 & & \mathsf{cFib}(\mathsf{U}, \Pi\mathsf{lem}, \ldots)\ \Xi\ (\lambda u.\Sigma v : \Gamma\ u.A\ u\ v) \\
 & & \to \mathsf{cFib}(\mathsf{U}, \Pi\mathsf{lem}, \ldots)\ \Xi\ \Gamma \\
\mathsf{ctx}\Pi_2\ \{\Xi\}\ \{\Gamma\}\ (A, \mathsf{fillA})\ (B, \mathsf{fillB}) & := & (\lambda u\ v.\Pi w : A\ u\ v.B\ (u, w)\ v, \\
 & & \quad \Pi\mathsf{lem}\ \Xi\ \Gamma\ A\ B\ \mathsf{fillA}\ \mathsf{fillB}) \\
 & & \\
\mathsf{ctxBdry} & : & \Pi\{\Xi\} : \mathcal{U}. \\
 & & \Pi\{\Gamma\} : \Xi \to \mathcal{U}. \\
 & & \Pi(A, \mathsf{fillA}) : \mathsf{cFib}(\ldots)\ \Xi\ \Gamma. \\
 & & \Pi\alpha : \Xi \to \mathsf{Cof} \\
 & & (\Pi u : \Xi.\Pi v : \Gamma\ u.[\alpha\ u] \to A\ u\ v) \\
 & & \to \mathsf{cFib}(\ldots)\ \Xi\ \Gamma \\
\mathsf{ctxBdry}\ \{\Xi\}\ \{\Gamma\}\ (A, \mathsf{fillA})\ \alpha\ a & := & (\lambda u\ v.A\ u\ v[\alpha\ u \mapsto a\ u\ v], \\
 & & \quad \mathsf{cfillBdry}(\Xi, \Gamma, A, \alpha, a, \mathsf{fillA})) \\
 & & \\
\mathsf{ctx}\Sigma & : & \Pi\{\Xi\} : \mathcal{U}. \\
 & & \Pi\{\Gamma\} : \Xi \to \mathcal{U}. \\
 & & \Pi(A, \mathsf{fillA}) : \mathsf{cFib}(\ldots)\ \Xi\ \Gamma. \\
 & & \mathsf{cFib}(\ldots)\ \Xi\ (\lambda u.\Sigma v : \Gamma\ u.A\ u\ v) \to \mathsf{cFib}(\ldots)\ \Xi\ \Gamma \\
\mathsf{ctx}\Sigma\ \{\Xi\}\ \{\Gamma\}\ (A, \mathsf{fillA})\ (B, \mathsf{fillB}) & := & (\lambda u\ v.\Sigma w : A\ u\ v.B\ u\ (v, w), \\
 & & \quad \mathsf{cfill}\Sigma(\Xi, \Gamma, A, B, \mathsf{fillA}, \mathsf{fillB})) \\
 & & \\
\mathsf{ctxU} & : & \Pi\{\Xi\} : \mathcal{U}. \\
 & & \Pi\{\Gamma\} : \Xi \to \mathcal{U}. \\
 & & (\Pi u : \Xi.\Gamma\ u \to \mathsf{Univ}(\ldots)) \to \mathsf{cFib}(\ldots)\ \Xi\ \Gamma \\
\mathsf{ctxU}\ \{\Xi\}\ \{\Gamma\}\ A & := & (\lambda u\ v.\mathsf{El}(A\ u\ v), \mathsf{cfillU}(\Xi, \Gamma, A))
\end{array}
$$

Figure 5.7: Combined notation for fiberwise filling closure conditions

$$
\begin{array}{l}
(\mathsf{ctx}\Pi\ (\lambda\_.\mathsf{Cof}) \\
(\mathsf{ctx}\Pi\ (\lambda\_.\mathbf{A}) \\
(\mathsf{ctx}\Pi\ (\lambda(\alpha, x).[\varphi\ x]) \\
(\mathsf{ctx}\Pi_2\ (\mathsf{ctx}\Pi\ (\lambda\_.\mathbf{B})\ (\mathsf{ctx}\Pi\ (\lambda(\alpha, x, \_, y).[\delta\ x\ y]) \\
\quad (\mathsf{ctxU}\ (\lambda(\alpha, x, \_, y, \_)\ A.(A\ y\ *))))) \\
(\mathsf{ctx}\Pi_2\ (\mathsf{ctx}\Pi\ (\lambda\_.\mathbf{B})\ (\mathsf{ctx}\Pi\ (\lambda(\alpha, x, \_, y).[(y = \beta\ y) \wedge \alpha]) \\
\quad (\mathsf{ctxBdry}\ (\mathsf{ctxU}\ (\lambda(\alpha, x, \_, y, \_)\ (A, b).A\ y\ *))\ (\lambda(\alpha, x, \_, y, \_).\delta\ x\ y) \\
\qquad (\lambda(\alpha, x, \_, y, \_)\ (A, b)\ \_.b\ y\ *)))) \\
(\mathsf{ctx}\Pi\ (\lambda\_.\mathbf{B}) \\
(\mathsf{ctx}\Pi\ (\lambda(\alpha, x, \_, z).[z = \beta\ x\ z]) \\
(\mathsf{ctxBdry}\ (\mathsf{ctxU}\ (\lambda(\alpha, x, \_, z, \_)\ (A, b, t).A\ z\ *))\ (\lambda(\alpha, x, \_, z, \_).\delta\ x\ z \vee \alpha) \\
\quad (\lambda(\alpha, x, \_, z, \_)\ (A, b, t)\ \_.b \vee t)))))))))
\end{array}
$$

Note that the parameters to cFill correspond to the filling problem the type satisfies, and not that described by the type.

The internal term corresponding to $\mathsf{Fill}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)$ follows from this by induction on the judgement $\mathsf{U}\ \mathsf{Univ}$.

If U is $\mathcal{U}$, none of this fibrancy proof is even needed, and $\mathsf{Fill}(\mathcal{U}, \ldots)$ is simply the type family itself.

$$
\begin{array}{rll}
\mathsf{Fill}(\mathcal{U}, \ldots) := \lambda A. & \Pi\alpha : \mathsf{Cof}. \\
& \Pi x : \mathbf{A}.\Pi * : [\varphi\ x]. \\
& \Pi b : \Pi y : \mathbf{B}.[\delta\ x\ y] \to A\ y\ *\,. \\
& \Pi t : \Pi y : \mathbf{B}.[(y = \beta\ y) \wedge \alpha] \to A\ y\ *\,[\delta\ x\ y \mapsto b]. \\
& \Pi y : \mathbf{B}.[y = \beta\ y] \to A\ y\ *\,[\delta\ x\ y \mapsto b, \alpha \mapsto t]
\end{array}
$$

If U is instead some $\mathsf{Univ}(\mathsf{U}', \Pi\mathsf{lem}', x : \mathbf{A}', y : \mathbf{B}', \varphi'\ x, \beta'\ y, \delta'\ x\ y)$, then by our inductive hypothesis assume we have defined a type family corresponding to the filling operation landing in U'.

$$
\cdot \vdash \mathsf{Fill}(\mathsf{U}', \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) : \Pi A : (\Pi y : \mathbf{B}.[y = \beta\ y] \to \mathsf{U}').\mathsf{U}'
$$

To construct the code landing in U, we must provide two things. First, the type family landing in U' is easily given as follows.

$$\cdot \vdash \lambda A.\mathsf{Fill}(\mathsf{U}', \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ (\mathsf{El}(\mathsf{U}', \ldots) \circ A) : \Pi A : (\Pi y : \mathbf{B}.[y = \beta\ y] \to \mathsf{U}).\mathsf{U}'$$

Second, we need to provide the filling operation, which is given by the second projection of $\mathsf{cFill}(\mathsf{U}', \Pi\mathsf{lem}', x : \mathbf{A}', y : \mathbf{B}', \varphi'\ x, \beta'\ y, \delta'\ x\ y)$ (applied to the unit term to correct for the empty contextual context). Combining these together, we define $\mathsf{Fill}(\mathcal{U}, \ldots)$.

$$\begin{aligned}
&\mathsf{Fill}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y) := \\
&\quad \mathsf{code}(\mathsf{U}', \Pi\mathsf{lem}', x : \mathbf{A}', y : \mathbf{B}', \varphi'\ x, \beta'\ y, \delta'\ x\ y) \\
&\qquad (\Pi y : \mathbf{B}.[y = \beta\ y] \to \mathsf{U}, \\
&\qquad \lambda A.\mathsf{Fill}(\mathsf{U}', \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)\ (\mathsf{El}(\mathsf{U}', \ldots) \circ A), \\
&\qquad ((\pi_2\ \mathsf{cFill}(\mathsf{U}', \Pi\mathsf{lem}', x : \mathbf{A}', y : \mathbf{B}', \varphi'\ x, \beta'\ y, \delta'\ x\ y))) *)
\end{aligned}$$

Using the same techniques, on can conclude that many of the fundamental building blocks used in type theories are always fibrant, such as path types, fibrant extension types and the contractible fill types. When working in type theories such at HoTT, cubical type theory, etc... the goal is to have the ability to discuss the additional structure introduced by modeling types as presheaves; these rules provide the foundational building blocks to safely describe the types that expose this categorical structure.

5.2.5 Semantics of Internally Definable Universes

While the rules interpret into the LOPS construction via the standard interpretation of the type theory into the topos $\widehat{\mathbb{C}}$, there are a number of tricks and nuances at play to make things work out as desired. To allow for the most general class of filling operations as defined in Definition 5.1.1 we semantically must construct each universe in a slice of the presheaf topos $\widehat{\mathbb{C}}$. We also have omitted any explicit reference to the $\flat$-modality from the syntax despite the $\flat$-modality being required to encode types with the LOPS construction; it "secretly" is taken into account by the syntactic rules, which we justify in this section.

For this section, let $\mathbb{U}$ represent the object classifier of the topos $\widehat{\mathbb{C}}$; in particular, we mean the the Grothendieck hierarchy of such objects, but leave the level annotations implicit. We also assume that $\mathbb{C}$ is closed under finite products, as is required by the LOPS construction.

Syntax	Semantic
$\mathcal{U}$	Grothendieck universe of topos
$\mathsf{Univ}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)$	LOPS construction over U
El_{U}	LOPS decode function for U
$\mathsf{El}(\mathsf{U}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)$	LOPS construction projection into U
$\mathsf{fill}(\mathsf{U}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)$	filler from LOPS
$\mathsf{Fill}(\mathsf{U}, \Pi\mathsf{lem}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)$	composition operation code in U
$\mathsf{code}(\mathsf{U}, x : \mathbf{A}, y : \mathbf{B}, \varphi\ x, \beta\ y, \delta\ x\ y)$	encode function from LOPS that "hides" the $\flat$-modality

Figure 5.8: Summary of topos semantics for internal LOPS universes

Definition 5.2.1. For this section, we define *filling data* to consist of a dependent tuple $(\mathbf{A}, \mathbf{B}, \varphi, \beta, \delta)$ where $\mathbf{A}$ and $\mathbf{B}$ are representables in $\widehat{\mathbb{C}}$, φ is a cofibration scoped by $\mathbf{A}$, δ is a cofibration scoped by $\mathbf{A} \times \mathbf{B}$, and β is an idempotent morphism from $\mathbf{A} \times \mathbf{B}$ to itself sliced over $\mathbf{A}$ via the first projection.

Definition 5.2.2 (from [47, Theorem 5.2])**.** A *universe object* is an object $\mathsf{U} \in \widehat{\mathbb{C}}$ such that, for some filling data $(\mathbf{A}, \mathbf{B}, \varphi, \beta, \delta)$, we have the following:

- a morphism $\mathsf{El}_{\mathsf{U}} \in \widehat{\mathbb{C}}(\mathsf{U}, \mathbb{U})$;
- a global element $\mathsf{fill}_{\mathsf{U}} \in \widehat{\mathcal{C}}(1, \mathsf{Fill}^{\mathbf{B}|\delta \hookrightarrow (\mathrm{id}=\beta)}_{\mathbf{A}|\varphi}\ \mathsf{U}\ \mathsf{El}_{\mathsf{U}})$;
- for every object Γ, morphism $A \in \widehat{\mathbb{C}}(\Gamma, \mathbb{U})$ and every global element $f \in \widehat{\mathcal{C}}(1, \mathsf{Fill}^{\mathbf{B}|\delta \hookrightarrow (\mathrm{id}=\beta)}_{\mathbf{A}|\varphi}\ \Gamma\ A)$ a morphism $\mathsf{code}_{\mathsf{U}}(A, f) \in \widehat{\mathbb{C}}(\Gamma, \mathsf{U})$.

These must also satisfy the equations below.

$\forall \Gamma \in \widehat{\mathbb{C}}, A \in \widehat{\mathbb{C}}(\Gamma, \mathbb{U}), f \in \widehat{\mathcal{C}}(1, \mathsf{Fill}^{\mathbf{B}|\delta \hookrightarrow (\mathrm{id}=\beta)}_{\mathbf{A}|\varphi}\ \Gamma\ A)$,

$$\begin{aligned} A &= \mathsf{El}_{\mathsf{U}} \circ \mathsf{code}_{\mathsf{U}}(A, f) \\ f &= \lambda x\ *\ p.\mathsf{fill}_{\mathsf{U}}\ x\ *\ (\lambda y\ *.\mathsf{code}_{\mathsf{U}}(A, f) \circ (p\ y\ *)) \end{aligned}$$

$\forall \Gamma \in \widehat{\mathbb{C}}, A \in \widehat{\mathbb{C}}(\Gamma, \mathsf{U})$,

$$A = \mathsf{code}_{\mathsf{U}}(\mathsf{El}_{\mathsf{U}} \circ A, \lambda x\ *\ p.\mathsf{fill}_{\mathsf{U}}\ x\ *\ (A \circ p\ x\ *))$$

Construction 5.2.3. Given filling data $(\mathbf{A}, \mathbf{B}, \varphi, \beta, \delta)$, one can constructively define a universe object $\mathsf{U}_{(\mathbf{A},\mathbf{B},\varphi,\beta,\delta)}$ in $\widehat{\mathcal{C}}$ classifying fibrations for the filling problem $\mathsf{Fill}^{\mathbf{B}|\delta\hookrightarrow(\mathrm{id}=\beta)}_{\mathbf{A}|\varphi}$.

Proof. We define the universe using the LOPS construction. The composition structure for LOPS is given as follows.

$$\begin{array}{lcl} C & : & \Pi A : (\Pi z : \mathbf{B}.[\![\beta\ z = z]\!]_{\mathsf{cof}} \to \mathbb{U}).\mathbb{U} \\ C\ A & := & \Pi \alpha : \mathsf{Cof}. \\ & & \Pi x : \mathbf{A}.\Pi * : [\varphi\ x]. \\ & & \Pi b : \Pi z : \mathbf{B}.[\![\delta\ x\ z]\!]_{\mathsf{cof}} \to A\ z\ *\,. \\ & & \Pi t : \Pi z : \mathbf{B}.[\![\beta\ z = z \wedge \alpha]\!]_{\mathsf{cof}} \to A\ z\ *\,[\delta\ x\ z \mapsto b]. \\ & & \Pi z : \mathbf{B}.[\![\beta\ z = z]\!]_{\mathsf{cof}} \to A\ z\ *\,[\delta\ x\ z \mapsto b, \alpha \mapsto t] \end{array}$$

For this composition structure to be compatible with LOPS, we must confirm the object $\Sigma\mathbf{B}.(\beta = \mathrm{id}_{\mathbf{B}})$ is tiny. This follows from the assumption $\mathbb{C}$ is closed under finite products, and that β is an idempotent morphism. From [15, Theorem 1], we know the category of presheaves over the idempotent completion of a category is equivalent to the category of presheaves over the original category. As $\Sigma\mathbf{B}.(\beta = \mathrm{id}_{\mathbf{B}})$ is a representable in the category of presheaves over the idempotent completion (and thus tiny), it is tiny in the original presheaf category. Given this holds and that the definition of C matches our intended notion of fibrancy, the LOPS construction gives us the desired universe. ⊠

5.2.6 Applications of Internally Definable Universes

The work in this chapter has been motivated by a few potential applications that are worthy of note. As has been mentioned throughout this chapter, we think that using this approach to define universes within a new a proof assistant will make implementation much easier and as an added bonus decrease the trusted code base, making the resulting proof assistant more reliable and less error-prone. As a second application, we believe that by making the definition of new classifying universes a simple task, one can consider doing so for more specialized situations and explore more niche classes of types than traditionally considered. In regards to this perspective, one example we find interesting would be considering the

universe of Kan types in Dedekind cubical type theory that additionally lift against the trivial cofibration enforcing a unique, weak total order on interval variables; as Dedekind cubical sheaves with respect to the total order are isomorphic to simplicial sets, we conjecture the fibrations classified by the universe of Kan Dedekind cubes that are weakly totally ordered should be Quillen equivalent to Kan simplicial spaces. The ability to so easily define such universes and explore universes for any number of (relatively-)arbitrary lifting problems side-by-side seems compelling and a particularly powerful proof assistant within which to explore mathematics.

Chapter 6

Implementation

The contents of this chapter are joint work with Daniel Licata. A summary of this chapter was presented at the 2nd International Conference on Homotopy Type Theory [82]. Reed Mullanix assisted with the implementation.

We implemented a portion of the type theory described in thesis building off of the cooltt proof assistant [1], and our implementation is available as a branch of the same repository [3]. To summarize the features of the implementation, its type theory contains the following features beyond those in "vanilla" cooltt (which is based off of Cartesian cubical type theory [7]):

- a second interval, $\mathbb{2}$—in addition to the homotopical interval $\mathbb{I}$—used to represent directed structure;
- an extended notion of cofibration built out of inequalities and equalities of terms and variables of *both* interval types, closed under conjunction, disjunction, universal quantification of either interval, top and bottom;
- a generalized notion of fibrant extension types, implementing and automating the contextual fibrancy reasoning described in Chapter 5. In particular, this makes the type

of the Cartesian filling problem an internally definable fibrant type, along with those corresponding to the filling problems we use in directed type theory.

The most notable omission from the current state of the implementation are any additional universes for the various new classes of types introduced in Chapter 3, and given in particular the lack of a universe of covariant fibrations directed cooltt also fails to support directed univalence (in its current state). That being said, combining the existing features one can use the current version of directed cooltt to internally define the filling problem style predicates used to define covariant fibrations, discrete fibrations, Segal types, and inner fibrations, and reason about the properties of such types (e.g. their closure conditions). We actually used directed cooltt ourselves to verify a few of the proofs appearing in Section 3.2!

6.1 Bicubical Directed Type Theory Without Connections

The most significant departure from the syntax given for the topos semantics of Chapter 3 is that we choose to omit connections for the directed interval in the implementation, and instead provide a builtin inequality cofibration (recall that the primary use of connections in the semantics was to define the inequality cofibration $x \leq y$ as the equality cofibration $x = x \sqcap y$ for directed interval variables x and y). It is important to note that, while the syntax of the implementation does not contain connection terms for the directed interval, it still can be interpreted into semantics containing connections (such as the semantics from Chapter 3), with the inequality cofibration interpreted as the equality above.

The algorithm for deciding cofibration propositions in the presence of connections is a Π_2^{P}-complete problem [66], while that for deciding cofibrations without connections but with inequalities is coNP-complete and thus much more tractable; as such, we sought to avoid this computational challenge by instead exploring how the surface syntax of the implementation could be modified to avoid connections without sacrificing the ability to express all that

one would hope to be able to encode in directed type theory. As a secondary benefit, this approach allows for the syntax to be interpreted into semantics based in more potential cube categories, as e.g. one can certainly imagine defining a new cube category containing the monomorphisms corresponding to inequality cofibrations without the additional morphisms generated from arbitrary connections.

6.1.1 Covariant Fibrations Without Connections

In order to account for the lack of connections for the directed interval, we had to update the notion of covariant filling (Definition 3.2.9). To summarize the problem before modifying the filling problem, recall that the filler is based on the idea of taking as input a term over a path p evaluated at $\mathbb{0}_2$, and outputting a term defined over the entire path. Should one need to, one could modify the path and instead provide $\lambda x.p\ (i \sqcap x)$ to instead provide as input a term over $p\ i$ and as output receive a term defined from i to $\mathbb{1}_2$, as can one do similar tricks with connections for other restrictions of the path. These tricks that leverage connections are fundamental and prove necessary in many proofs, including those appearing in this thesis.[1]

Thankfully, one can provide a new definition for covariant filling that generalizes all of these tricks, and furthermore is provably equivalent to the "standard" definition from Chapter 3 in the presence of connections.

Definition 6.1.1. A *relativized covariant filling structure* on $A : \Gamma \to \mathsf{Type}$ is a term of type $\mathsf{covFill}_\Gamma\ A$:

[1] e.g. proving that $\mathsf{U}_{\mathsf{cov}}$ is closed with respect to Σ-types. Note that the corresponding proof using the new notion of covariance defined here appears at the end of Section 6.2.

$$
\begin{aligned}
&\mathsf{rcovFill}^{2}\ (i\ j : 2)\ (* : [\![i \leq j]\!]_{\mathsf{cof}}) \\
&\qquad (A : \Pi z : 2.[\![i \leq z \leq j]\!]_{\mathsf{cof}} \to \mathsf{Type}) := \Pi\alpha{:}\Omega_{\mathrm{cof}}. \\
&\qquad\qquad \Pi t{:}(\Pi z{:}2.[\![i \ \leq z \leq j]\!]_{\mathsf{cof}} \\
&\qquad\qquad\qquad \to [\![\alpha]\!]_{\mathsf{cof}} \to A\ z\ *). \\
&\qquad\qquad \Pi b{:}(A\ i\ *)[\alpha \mapsto t\ i\ *]. \\
&\qquad\qquad (A\ j\ *)[\alpha \mapsto t\ j\ *, i = j \mapsto b] \\
&\mathsf{rcovFill}_{\Gamma}\ (A : \Gamma \to \mathsf{Type}) := \qquad \Pi i\ j : 2.\Pi * : [\![i \leq j]\!]_{\mathsf{cof}}. \\
&\qquad\qquad \Pi p{:}(\Pi z : 2.[\![i \leq z \leq j]\!]_{\mathsf{cof}} \to \Gamma). \\
&\qquad\qquad \mathsf{rcovFill}^{2}\ i\ j\ *\ (A \circ p)
\end{aligned}
$$

In the presence of a witness $\mathsf{rcovA} : \mathsf{rcovFill}_{\Gamma}\ A$, we write $\mathsf{cov}_{A}^{z:i\to j}\ p\ [\alpha \mapsto t\ z]\ b$ to denote $\mathsf{rcovA}\ i\ j\ *\ p\ \alpha\ (\lambda z.t\ z)\ b$.

Let us quickly demonstrate how this notion of covariance coincides with that given in Definition 3.2.9 in the presence of connections. For our purposes here we simply will demonstrate their interderivability, and not formally construct the entire equivalence. As such, assume a context type Γ and type family $A : \Gamma \to \mathsf{Type}$. Beginning with a witness to $\mathsf{rcovFill}_{\Gamma}\ A$, it is particularly straightforward to derive a solution to the standard covariant filling problem.

$$
\begin{aligned}
&\mathsf{covA} &&: \mathsf{covFill}_{\Gamma}\ A \\
&\mathsf{covA}\ p\ \alpha\ t\ b\ k &&:= \mathsf{cov}_{A}^{z:\mathbb{0}_2\to k}\ p\ [\alpha \mapsto t\ z]\ b
\end{aligned}
$$

By instantiating the relativized covariant filling problem such that i is $\mathbb{0}_2$ and j is k, one recovers the original notion of covariance. To derive a solution $\mathsf{rcovA} : \mathsf{rcovFill}_{\Gamma}\ A$ from a witness to the standard problem, one instead modifies the path supplied to the filling problem using connections.

$$
\begin{aligned}
&\mathsf{rcovA} &&: \mathsf{rcovFill}_{\Gamma}\ A \\
&\mathsf{rcovA}\ p\ i\ j\ *\ \alpha\ t\ b &&:= \mathsf{cov}_{A}^{z:\mathbb{0}_2\to j}\ (\lambda z.p(j \sqcap (i \sqcup z)))\ [\ \alpha \mapsto t\ (j \sqcap (i \sqcup z))\ *, \\
&&&\qquad\qquad i = j \mapsto b]\ b
\end{aligned}
$$

In this case, it is important to note how the types of t, b and the output interact with the modification to the path p. First, as $t : \Pi z{:}2.[\![i \leq z \leq j \sqcap \alpha]\!]_{\mathsf{cof}} \to A\ (p\ z)$, when instantiated

at $j \sqcap (i \sqcup z)$, the resulting term has type $[\![i \leq j \sqcap (i \sqcup z) \leq j]\!]_{\mathsf{cof}} \to [\![\alpha]\!]_{\mathsf{cof}} \to A\ (p\ (j \sqcap (i \sqcup z)))$. By the equational rules for connections (Figure 3.3), $[\![i \leq j \sqcap (i \sqcup z) \leq j]\!]_{\mathsf{cof}}$ is always true and thus we can apply the term to the witness of such resulting in $t\ (j \sqcap (i \sqcup z))\ *$ which has type $[\![\alpha]\!]_{\mathsf{cof}} \to A\ (p\ (j \sqcap (i \sqcup z)))$ as expected by the standard filling operation. The term b has type $(A\ (p\ i))[\alpha \mapsto t\ i\ *]$, which equals the type $A\ (p\ (j \sqcap (i \sqcup \mathbb{O}_2)))[\alpha \mapsto t\ (j \sqcap (i \sqcup \mathbb{O}_2))\ *]$ by again using the equational rules for connections (which in particular is the type expected by the standard filling problem where the term b is utilized). Lastly, the type expected of $\mathsf{cov}_A^{z:\mathbb{O}_2 \to j}\ (\lambda z.p(j \sqcap (i \sqcup z)))\ [\alpha \mapsto t\ (j \sqcap (i \sqcup z))\ *, i = j \mapsto b]\ b$ for the above term to type check is $(A\ (p\ j))[\alpha \mapsto t\ j\ *, i = j \mapsto b]$. By directly computing its type from the definition, we instead see it has type $(A\ (p\ (j \sqcap (i \sqcup j))))[\alpha \mapsto t\ (j \sqcap (i \sqcup j))\ *, i = j \mapsto b]$; however, this type is scoped by a witness to $i \leq j$, and thus by the equations for connections $j \sqcap (i \sqcup j)$ reduces to j, and thus the above definition of rcovA is well-typed.

While we have not added the universe of relativized covariant types to directed cooltt, the plan to do so is as follows: Despite not fitting into the schema of fibrant filling problems defined in Chapter 5, the type of relativized covariant fillers is Kan fibrant and compatible with LOPS universe semantics and as such we intend to manually add the universe incorporating as much of the approach described in Section 5.2.3 as is possible. While the universe itself must be manually added to the syntax of the language, we then plan to add a code constructor to inhabit the universe that takes as input a cooltt type along with an internally defined solution to the filling problem in the same way as the universes are inhabited in Section 5.2.3.

6.2 Sample Code in Directed CoolTT

We now present some sample code written in directed cooltt.

As a starting point, let us look at the definition of the directed path type.

```
def hom0 (A : (i : 2) → type)
         (a : A d0)
```

```
        (b : A d1) : type :=
    (i : 2) → fsub A i with [i = d0 ⇒ a | i = d1 ⇒ b]
```

While relatively straightforward and similar to definitions of path types in the current implementations of cubical type theory, one key syntactic element likely stands out as unusual: the `fsub` type former. In cooltt, the term "extension type" refers to RS extension types (Definition 2.3.11), i.e. a type with a boundary restriction directly wrapped with a Π-type that binds all interval variables appearing in the boundary restriction. To refer to a "normal" extension type consisting of just the type with a boundary restriction, cooltt uses the term "sub type." To keep things consistent, directed cooltt introduces the `fsub` type former to represent sub types that additionally are required to be used in a fibrant way according to the rules explored in Chapter 5. Given this is the case, an `fsub` type is always a subterm of a type that is being checked as a term of the universe of Kan types (which is denoted in the language by `type`).

As the next code sample, consider the definition of the inner filling problem.

```
def isInner (A : (i j : 2) → <i ≤ j> → type) : type :=
  (α : cof) →
  (b : (i j : 2) → <i=d0 ∨ j=d1> → A i j) →
  (t : (i j : 2) → <i ≤ j ∧ α>
     → fsub A i j with [i=d0 ∨ j=d1 ⇒ b i j]) →
  (i j : 2) → <i ≤ j> →
  fsub A i j with [ α            ⇒ t i j
                  | i=d0 ∨ j=d1 ⇒ b i j ]

def relInner (Γ : type) (A : Γ → type) : type :=
    (p : (i j : 2) → <i ≤ j> → Γ)
    → isInner {i j ⇒ {A {p i j}}}
```

Conveniently, the code looks nearly identical to that which we wrote manually in Definition 3.2.18, but what is important to realize is that, as the fiber-wise contextual reasoning is automated, `isInner` and `relInner` are automatically fibrations landing in `type`, the universe of Kan fibrations.

We also used directed cooltt to verify a few of the proofs appearing in Chapter 3. First, we have the definition of the inner filling operation for Σ-types...

```
def sigmaInner (Γ : type)
               (A : Γ → type)
               (B : (γ : Γ) → A γ → type)
               (inA : relInner Γ A)
               (inB : relInner {(γ : Γ) × A γ}
                               {p ⇒ B {fst p} {snd p}})
               : relInner Γ {γ ⇒ (a : A γ) × B γ a} :=
    p α b t i j ⇒
        let a : (i j : 2) → <i ≤ j> → A {p i j} :=
            inA p α {i j ⇒ fst {b i j}}
                    {i j ⇒ fst {t i j}} in
        [ a i j
        , inB {i j ⇒ [p i j , a i j ]} α
              {i j ⇒ snd {b i j}}
              {i j ⇒ snd {t i j}} i j ]
```

...and here is the definition of the inner filling operation for directed path types.

```
def HomInner (Γ : type)
             (A : Γ → 2 → type)
             (inA : (i : 2) → relInner {Γ} {p ⇒ A p i})
             (a0 : (γ : Γ) → A γ d0) (a1 : (γ : Γ) → A γ d1)
             : relInner Γ {γ ⇒ hom0 {A γ} {a0 γ} {a1 γ}} :=
    p α b t i j x ⇒ inA x p
                        {α ∨ x = d0 ∨ x = d1}
                        {i j ⇒ b i j x}
                        {i j ⇒ [α ⇒ t i j x
                                | x = d0 ⇒ a0 {p i j}
                                | x = d1 ⇒ a1 {p i j}]}
                        i j
```

Both constructions appear within Theorem 3.4.2.

Lastly, let's look at a proof that uses the relativized covariance definition. First, we define the type of relativized filling structures.

```
def relCov (Γ : type) (A : Γ → type) : type :=
    (i j : 2) → <i ≤ j> →
    (p : (k : 2) → <i ≤ k ∧ k ≤ j> → Γ) →
    (α : cof) →
    (b : A {p i}) →
    (t : (k : 2) → <i ≤ k ∧ k ≤ j ∧ α>
       → fsub A {p k} with [i=k ⇒ b]) →
    fsub A {p j} with [ α  ⇒ t j
                      | i=j ⇒ b ]
```

The proof that relativized covariant types are closed under Σ-types follows the same pattern as for inner fibrations: One simply pairs together the two solutions.

```
def sigmaCov (Γ : type) (A : Γ → type)
             (B : (γ : Γ) → A γ → type)
             (covA : relCov Γ A)
             (covB : relCov {(γ : Γ) × A γ}
                            {p ⇒ B {fst p} {snd p}})
             : relCov Γ {γ ⇒ (a : A γ) × B γ a} :=
    i j p α b t ⇒
        let a : (k : 2) → [i ≤ k ∧ k ≤ j] → A {p k} :=
            k ⇒ covA i k p α {fst b} {k ⇒ fst {t k}} in
        [ a j
        , covB i j {k ⇒ [p k , a k]} α
                   {snd b} {k ⇒ snd {t k}} ]
```

Chapter 7

Future Work

An exciting secondary outcome of this body of work in bicubical directed type theory is that it has layed the foundation for a sizable amount of potential future research. While the projects introduced in this chapter are far from forming an exhaustive list, here are a few of the key directions we are most enthusiastic to explore.

7.1 Directed Higher Inductive Types

In the presence of directed univalence, one would additionally hope for a theory containing directed higher inductive types (DHITs). These types allow for one to inductively define types containing directed structure, and in conjunction with directed univalence allow for novel proof techniques in verification; in particular, the combination allows one to directly formalize functorial semantics using the internal categorical structure of types. We intend to introduce a schema for DHITs based off of the schema for higher inductive types in cubical type theory by Cavallo and Harper [19].

7.2 More Implementation(s)

While much of the work in this thesis is already implemented in directed cooltt [3], we would like to ultimately implement a complete working version of directed type theory as a proof assistant. Specifically, this involves using internal LOPS universes as described in Chapter 5 to add universes classifying the inner and covariant types, adding inductive and directed higher inductive types into the implementation, and finally taking into account the cobar modal theory from Chapter 4 so that the proof assistant supports computational directed univalence.

7.2.1 Computational Univalence and Challenge of Cobar

While the list above mostly provides a straightforward summary of the future we see for directed cooltt, one particular aspect warrants deeper discussion: incorporating cobar-modal types into the syntax and computational semantics. While the construction given in Chapter 4 is fully constructive, it is not immediately clear how the addition of the cobar modality and the resulting constructions built using it behave in the setting of a programming language/proof assistant. The key challenge is that, when doing constructive mathematics, one has the ability to see "outside" of the type theory, and is working from a global perspective; when defining a syntax and computational model for a programming language, nearly all constructions and rules are defined locally with respect to an arbitrary context, and furthermore they must be defined in such a way that they are stable under context substitutions. Unfortunately, the cobar construction itself is particularly global in nature: It initially is defined piecewise at every dimension (i.e. object) of the directed cubes individually, and the construction then adheres in such a way that the individual components can be combined to ultimately result in a definition that is coherent over all of the directed cubes. To work with the construction, one must know at which directed dimension the presheaves are evaluated at, and then work from that specific dimension's definition. While this is a perfectly reason-

able way to work in the categorical semantics, it does not translate to the local, open-context nature of computational rules. Identifying how the cobar construction translates to a computational setting therefore warrants its own research project, and only with its solution will we have a *computational* notion of directed univalence.

7.2.2 Internal LOPS Universes

As mentioned in Section 5.2.6, we are also interested in implementing a proof assistant for bicubical type theory that is built from the ground up to incorporate internally definable LOPS universes.

7.3 Applications in Verification

Upon implementing a fully-functional proof assistant directed type theory, a potentially fruitful body of work becomes possible in exploring how new directed techniques can be developed to aid software verification.

7.3.1 Functorial Semantics

As introduced in Section 1.1, we believe directed type theory has the potential to be particularly useful for software verification by opening up the ability to naturally encode functorial semantics within the proof assistant. Combining the fact that all constructions are enforced to be functorial, and that directed type theory has the ability to "automatically" identify when many constructions are natural, designing one's formalization to leverage these properties should significantly simplify both definitions and proofs; furthermore, a large portion of the existing body of work formally studying software and programming languages already takes advantage of these properties of category theory on paper, demonstrating both how to structure a functorial approach to formal verification and the many benefits the categorical approach provides mathematically.

7.3.2 The Structure Preservation Principle

Considering a more specific application, one technique we wish to investigate specifically is whats we call the structure preservation principle (SPP). In a nutshell, SPP is the directed analogue of the structure identity principle (SIP): the idea that the mathematical properties of structures should be fully invariant with respect to isomorphisms of the structures. From the perspective of software verification and programming language theory, this corresponds to the "free theorems" granted by parametricity arguments; these arguments demonstrate that two implementations of the same abstract interface are fully interchangeable without effecting the behavior of programs that use the interface so long as each individual component provided to the interface from each implementation behaves identically. Angiuli, Cavallo, Mörtberg and Zeuner demonstrated how cubical type theory allows one to use SIP to derive such theorems within cubical type theory [9], and from it we asked the natural followup question: How would these ideas translate to directed type theory? From this, we landed on the structure preservation principle: mathematical properties of structures should be fully preserved with respect to homomorphisms of the structures. In terms of verification, this corresponds to being able to swap out one implementation of an abstract interface for another so long as there is a function from the first to the second that preserves the behavior of each component provided to the interface by the implementations. While certainly this can apply across a number of situations arising in software verification, most notable this pattern aligns with the structure of compiler correctness proofs.

When proving a compiler correct, one generally considers a function translating abstract syntax of one language to another, and alongside each languages' abstract syntax are a number of relations defined over the syntax specifying the behavior of that language, e.g. how it computes, which terms are values, etc.... For a compiler to be proven correct, one then shows that the compiling function preserves the behavior of programs, which specifically corresponds to showing that some theorem built from these abstract relations is preserved. Using SPP in directed type theory, one should be able to expedite this process by working

more locally to show the individual relations are preserved by the compiling function, and in doing so conclude the final theorem "for free." As an example, let us demonstrate how this works by proving that type erasure for the simply typed lambda calculus preserves the property of program termination.

To begin the example we first define the abstract structure used to represent a syntax. The idea is that a syntax consists with a type of terms `Syn` paired with a binary small-step relation `_ ⇒ _` and a unary value relation `Val`. In addition, we will also need to abstract structure to carry around a term of the syntax as well, and we will see why that is necessary by the end of the example. Putting this all together, we define the structure as the term `S` below.

```
S : UCov → UInner
S = λSyn.(_⇒_ : Syn → Syn → UCov)×(Val : Syn → UCov)×Syn
```

Note here that the type family specifying the abstract structure lands in `UInner`. For this technique there is no need for the stronger restriction that `S` land in the covariant universe; furthermore, we explicitly do not want such a restriction in this case as it would not allow us to write down the types of the relations, as they use covariant types in contravariant positions.

As we desire to prove that type erasure preserves termination, we will need a definition of the multistep relation. We do so abstractly over the syntax structure in the following definition.

```
data _⇒*_ {Syn : UCov}
          {(_⇒_ , Val) : S Syn}
          (t t' : Syn) : UCov where
  srefl  : Path t t' → t ⇒* t'
  strans : (t'' : Syn) → t ⇒* t'' → t'' ⇒ t' → t ⇒* t'
```

At this point, we have all of the pieces needed to define the proposition `P` classifying termination. Given some syntax, small step relation, value relation, and term `t`, the term `t` terminates if we can identify a term `t'` such that `t` multisteps to `t'`, and `t'` is a value.

```
P : (Σ C:UCov.S C) → UCov
P = λ(Syn, _⇒_, Val, t).Σ t':Syn.(t ⇒* t' × Val t')
```

This predicate identifies the singular case when the term `t` provided as part of the abstract structure terminates; note that were one to remove `t` from the structure and instead quantify over all terms with a Π as part of the predicate type itself, the predicate would no longer land in `UCov` as the Π would quantify over a covariant type in a contravariant position; furthermore, we need the predicate to land in `UCov` in order to leverage directed composition for the final result. By the end of this example, we will see that this limitation is a good thing, as without it one could prove results that are obviously incorrect.

Now assume we have defined types corresponding to the syntax of the simply typed lambda calculus and the untyped lambda calculus, along with predicates for their small step operational semantics and their value relation...

```
λt:STLC.(_⇒STLC_ , ValSTLC , t) : STLC → S STLC
λt:LC.(_⇒LC_ , ValLC , t) : LC → S LC
```

...and we have defined the type erasure function.

```
erase : STLC → LC
```

The meat of the proof consists of two lemmas: type erasure preserves the small step relation, and it preserves the value relation.

```
rel⇒ : Πt t':STLC.t ⇒STLC t' → (erase t) ⇒LC (erase t')
relVal : Πt:STLC.ValSTLC t → ValLC (erase t)
```

After proving both lemmas, the two can be combined to define a dependent morphism in `S` over the morphism in `UCov` given by directed univalence applied to `erase` that begins at the `STLC` structure and ends at the `LC` structure.

```
rel : Πt:STLC.HomO (dua erase)
                   (STLC, _⇒STLC_ , ValSTLC , t)
                   (LC  , _⇒LC_ , ValLC , erase t)
```

The second fundamental proof we need is that every term in the simply typed lambda calculus terminates. Using the previously defined components, this corresponds to providing

a term of type Πt : STLC.P (STLC, _ $\Rightarrow_{\text{STLC}}$ _, ValSTLC, t), which unfolds to the expected type.

```
pSTLC : Πt:STLC.P (STLC, _⇒STLC_ , ValSTLC , t)
pSTLC : Πt:STLC.Σ t':STLC.(t ⇒*STLC t' × ValSTLC t')
```

We now arrive at the payoff: Noting that the predicate P lands in UCov, we have access to directed transport and thus we can transport the proof that all terms in the simply typed lambda calculus terminate to one indicating that any untyped lambda calculus term that is *in the image of type erasure* also terminates.

```
pLC : Πt:STLC.P (LC , _⇒LC_ , ValLC , erase t)
pLC : Πt:STLC.Σ t':LC.((erase t) ⇒*LC t' × ValLC t')
pLC = λt.dtransp P (Σhom (dua erase, rel t)) (pSTLC t)
```

Recalling back to the beginning when we defined S and P, we needed to include a term as a part of the structure tuple with the predicate describing when that individual term terminated. Defining things in this way was required so that P landed in the covariant universe. Now at the end of this example, we see that doing so resulted in the final theorem concluding that all untyped lambda terms terminate so long as they are in the image of type erasure for some typed lambda term, which is indeed a true statement. Were the predicate P defined to express that all terms terminate, the proof concluded by using directed transport as done above would say that all untyped lambda terms terminate, which we know to be false; thankfully, such a predicate P is not properly typed as the type family is not covariant, and thus one cannot prove the false statement.

7.4 Model Categorical Semantics

Shifting to the mathematical applications of this work, those using proof assistants for synthetic mathematics care greatly about the mathematical semantics of the theory in which they do their work; as such, we hope to formally develop the model category semantics of bicubical directed type theory. While this is still very much a conjecture and future work,

here we sketch out the skeleton of what hopefully will become the proof, along with our reasoning for why we believe this to be the case.

While being relatively agnostic to our base model structure on cubical sets, let's assume it is defined in the "standard" way, being cofibrantly generated by pushout products of the endpoint inclusion with the boundary maps of the cube.

Now, we lift this filling problem to bicubical sets (the additional cube category being the Dedekind cubes) by using the same generators except our generating cofibrations used in the pushout product now include boundary maps of both the normal cubes and the directed cubes. Having done this, we will now define our fibrant objects to be those that are "fibrant" with respect to the lifted filling problem, and are also cobar modal. We conjecture this should give us the injective fibrations in bicubical sets over the underlying model structure on cubical sets via Lemma 8.22 in [71]. We already have shown that weak equivalences of cobar-modal types are level-wise weak equivalences of cubical sets in Section 4.2.5, and thus as both the fibrations and weak equivalences coincide, this model structure should indeed be the injective model structure on bicubical sets.

Bibliography

[1] cooltt. https://github.com/RedPRL/cooltt. 171

[2] coq. https://coq.inria.fr. 49

[3] directed cooltt. https://github.com/RedPRL/cooltt/tree/directed-2.0. 19, 171, 180

[4] Rzk. https://github.com/rzk-lang/rzk. 21

[5] Simplicial HoTT and synthetic ∞-categories in rzk. https://github.com/rzk-lang/sHoTT. 21

[6] Yoneda for ∞-categories in rzk. https://github.com/emilyriehl/yoneda. 21

[7] Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. Syntax and models of cartesian cubical type theory. *Mathematical Structures in Computer Science*, 31:424–468, 2021. 1, 54, 59, 62, 64, 65, 76, 81, 83, 86, 99, 104, 107, 171

[8] Carlo Angiuli, Evan Cavallo, Kuen-Bang Hou, Robert Harper, and Jonathan Sterling. The redprl proof assistant (invited paper). In *Proceedings of the 13th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, pages 1–10, 2018. 15

[9] Carlo Angiuli, Evan Cavallo, Anders Mörtberg, and Max Zeuner. Internalizing representation independence with univalence. *Proceedings of the ACM on Programming Languages*, 5(POPL):1–30, 2021. 182

[10] Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. Cartesian cubical computational type theory: Constructive reasoning with paths and equalities. In *Computer Science Logic*, 2018. 1, 54

[11] S. Awodey and M. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 2009. 1, 23

[12] Steve Awodey. A Quillen model structure on the category of cartesian cubical sets. Talk at the 1st International Conference on Homotopy Type Theory 2019, available from https://hott.github.io/HoTT-2019, 2019. 136

[13] Marc Bezem, Thierry Coquand, and Simon Huber. A model of type theory in cubical sets. Preprint, September 2013. 1, 54

[14] Lars Birkedal, Ales Bizjak, Ranald Clouston, Hans Bugge Grathwohl, Bas Spitters, and Andrea Vezzosi. Guarded cubical type theory: Path equality for guarded recursion. arXiv:1606.05223, 2016. 17

[15] Francis Borceux and Dominique Dejean. Cauchy completion in category theory. *Cahiers de topologie et géométrie différentielle catégoriques*, 27(2):133–146, 1986. 83, 169

[16] Ulrik Buchholtz and Edward Morehouse. Varieties of cubical sets. In *Relational and Algebraic Methods in Computer Science: 16th International Conference, RAMiCS 2017, Lyon, France, May 15-18, 2017, Proceedings*. 2017. 56, 80, 82

[17] Ulrik Buchholtz and Jonathan Weinberger. Type-theoretic modalities for synthetic $(\infty, 1)$-categories. Talk at the 1st International Conference on Homotopy Type Theory 2019, 2019. 20, 79

[18] Ulrik Buchholtz and Jonathan Weinberger. Synthetic fibered $(\infty, 1)$-category theory. *Higher Structures*, 7(1):74–165, 2023. 20

[19] Evan Cavallo and Robert Harper. Higher inductive types in cubical computational type theory. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2019. 3, 179

[20] Evan Cavallo and Robert Harper. Internal parametricity for cubical type theory. *Logical Methods in Computer Science*, 17(4):5:1–5:60, 2021. 20

[21] Evan Cavallo, Emily Riehl, and Christian Sattler. On the directed univalence axiom, 2018. Talk at the AMS Special Session on Homotopy Type Theory, Joint Mathematics Meetings. 17, 20, 69, 74, 75

[22] Evan Cavallo and Christian Sattler. Relative elegance and cartesian cubes with one connection, 2023. 60

[23] Denis-Charles Cisinski. *Les préfaisceaux comme modèles des types d'homotopie*. Société mathématique de France, 2006. 59

[24] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. In T. Uustalu, editor, *Post-proceedings of the 21st International Conference on Types for Proofs and Programs (TYPES 2015)*, pages 5:1–5:34, 2018. doi: 10.4230/LIPIcs.TYPES.2015.5. 1, 20, 54, 61, 65, 67, 79, 86, 89, 93, 95, 102, 103, 104, 106, 107

[25] Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the NuPRL Proof Development System*. Prentice Hall, 1986. 15

[26] Thierry Coquand. Constructive presheaf models of univalence. Talk at the 1st International Conference on Homotopy Type Theory 2019, 2019. 18

[27] Thierry Coquand, Simon Huber, and Anders Mörtberg. On higher inductive types in cubical type theory. In *IEEE Symposium on Logic in Computer Science*, 2018. 64

[28] Thierry Coquand, Fabian Ruch, and Christian Sattler. Constructive sheaf models of type theory. *Mathematical Structures in Computer Science*, 31(9):979–1002, 2021. iv, 18, 114, 116, 117, 118, 122, 123, 124, 127, 136, 139

[29] Thierry Coquand and Christian Sattler. A model structure on some presheaf categories. Available from cse.chalmers.se/~coquand/mod2.pdf, 2016. 136

[30] Lisbeth Fajstrup, Eric Goubault, Emmanuel Haucourt, Samuel Mimram, and Martin Raussen. *Directed algebraic topology and concurrency*. Springer, 2016. 2

[31] Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *IEEE Symposium on Logic in Computer Science*, 1999. 2

[32] Peter Freyd. Aspects of topoi. *Bulletin of the Australian Mathematical Society*, 7(1):1–76, 1972. 44, 45

[33] Nicola Gambino and Richard Garner. The identity type weak factorisation system. *Theoretical Computer Science*, 409(3):94–109, 2008. 23

[34] Nicola Gambino and Simon Henry. Towards a constructive simplicial model of univalent foundations. https://arxiv.org/abs/1905.06281, 2019. 16

[35] Richard Garner. Two-dimensional models of type theory. *Mathematical. Structures in Computer Science*, 19(4):687–736, 2009. 23

[36] David Gepner and Joachim Kock. Univalence in locally cartesian closed ∞-categories. In *Forum Mathematicum*, volume 29, pages 617–652. De Gruyter, 2017. 23

[37] Marco Grandis. *Directed Algebraic Topology: Models of non-reversible worlds*. Cambridge University Press, 2009. 2

[38] Jason Jonathan Hickey. *The MetaPRL logical programming environment*. PhD thesis, Cornell University, January 2001. 15

[39] Martin Hofmann. On the interpretation of type theory in locally cartesian closed categories. In *International Workshop on Computer Science Logic*, pages 427–441. Springer, 1994. 46

[40] Martin Hofmann. *Extensional Concepts in Intensional Type Theory*. PhD thesis, University of Edinburgh, 1995. 15, 17, 23, 61

[41] Martin Hofmann. Semantical analysis of higher-order abstract syntax. In *IEEE Symposium on Logic in Computer Science*, 1999. 2

[42] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory*. Oxford University Press, 1998. 23

[43] Daniel M Kan. Abstract homotopy. *Proceedings of the National Academy of Sciences*, 41(12):1092–1096, 1955. 59

[44] Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. The simplicial model of univalent foundations. arXiv:1211.2851, 2012. 46

[45] Daniel R. Licata and Robert Harper. 2-dimensional directed type theory. In *Mathematical Foundations of Programming Semantics (MFPS)*, 2011. 2, 16

[46] Daniel R. Licata and Robert Harper. Canonicity for 2-dimensional type theory. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2012. 23

[47] Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. Internal universes in models of homotopy type theory. In *International Conference on Formal Structures for Computation and Deduction*, 2018. 17, 18, 20, 55, 64, 77, 80, 82, 99, 100, 104, 144, 153, 160, 161, 168

[48] Peter LeFanu Lumsdaine. Weak ω-categories from intensional type theory. In *International Conference on Typed Lambda Calculi and Applications*, 2009. 23

[49] Peter LeFanu Lumsdaine and Michael A. Warren. The local universes model: an overlooked coherence construction for dependent type theories. *ACM Transactions on Computational Logic (TOCL)*, 16(3):1–31, 2015. 46

[50] Saunders MacLane and Ieke Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer Science & Business Media, 1992. 48

[51] P. Martin-Löf. Constructive mathematics and computer programming. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 312(1522):501–518, 1984. 15

[52] Paige Randall North. Towards a directed homotopy type theory. arXiv:1807.10566, 2018. 2, 16

[53] Paige Randall North. Towards a directed homotopy type theory. *Electronic Notes in Theoretical Computer Science*, 347:223–239, 2019. 20

[54] Andreas Nuyts. Towards a directed HoTT based on 4 kinds of variance. Master's thesis, KU Leuven, 2015. 16

[55] Andreas Nuyts. A model of parametric dependent type theory in Bridge/Path cubical sets. arXiv:1706.04383, 2017. 20

[56] Andreas Nuyts. Contributions to multimode and presheaf type theory. *eng. PhD thesis. KU Leuven*, 2020. 21, 144

[57] Ian Orton and Andrew M. Pitts. Axioms for modelling cubical type theory in a topos. In *Computer Science Logic*, pages 24:1–24:19, 2016. 17, 52, 62, 76, 93

[58] Ian Orton and Andrew M. Pitts. Axioms for modelling cubical type theory in a topos. *Logical Methods in Computer Science*, 14(4:23):1–33, December 2018. Special issue for CSL 2016. iii, 17, 52, 62, 76, 81

[59] Kevin Quirin. *Lawvere-tierney sheafification in homotopy type theory*. PhD thesis, Ecole des Mines de Nantes, 2016. 124

[60] Emily Riehl. *Categorical Homotopy Theory*. Cambridge University Press, 2014. 18

[61] Emily Riehl. *Category theory in context*. Courier Dover Publications, 2017. 25

[62] Emily Riehl and Michael Shulman. A type theory for synthetic ∞-categories. *Higher Structures*, 1(1):116–193, 2017. iii, iv, 2, 5, 16, 18, 21, 67, 72, 77, 78, 85, 86, 87, 92, 93, 95, 96, 97, 98, 100, 131

[63] Emily Riehl and Dominic Verity. The theory and practice of Reedy categories. *Theory and Applications of Categories*, 29(9), 2014. 18

[64] Egbert Rijke. Introduction to homotopy type theory, 2022. https://arxiv.org/abs/2212.11082. 22, 23

[65] Egbert Rijke, Michael Shulman, and Bas Spitters. Modalities in homotopy type theory. *Logical Methods in Computer Science*, 16(1):2:1–2:79, 2020. 124

[66] Robert Rose, Daniel R. Licata, and Matthew Z. Weaver. Deciding entailment for cofibration languages. Talk at the 2nd International Conference on Homotopy Type Theory, available from https://hott.github.io/HoTT-2023/slides/rose.pdf/, 2023. 172

[67] Christian Sattler. Do cubical models of type theory also model homotopy types? Talk at Workshop on Types, Homotopy Type theory, and Verification, Hausdorff Institute for Mathematics, 2018. 18

[68] Christian Sattler and Emily Riehl. Directed univalence for the left fibration classifier. Private correspondence, 2018. 17, 20, 69, 74, 75, 137

[69] Robert A.G. Seely. Locally cartesian closed categories and type theory. In *Mathematical proceedings of the Cambridge philosophical society*, volume 95, pages 33–48. Cambridge University Press, 1984. 44, 45

[70] Michael Shulman. Homotopy limits and colimits and enriched homotopy theory. https://arxiv.org/abs/math/0610194, 2009. 18

[71] Michael Shulman. All $(\infty, 1)$-toposes have strict univalent universes. 2019. https://arxiv.org/abs/1904.07004. 18, 186

[72] Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. A cubical language for bishop sets. *Logical Methods in Computer Science*, 18, 2022. 15

[73] Thomas Striecher and Jonathan Weinberger. Simplicial sets inside cubical sets. *Theory and Applications of Categories*, 37(10):276–286, 2021. 79

[74] The Univalent Foundations Program, Institute for Advanced Study. *Homotopy Type Theory: Univalent Foundations Of Mathematics*. Available from homotopytypetheory.org/book, 2013. 1, 22, 23, 61

[75] Benno van den Berg and Richard Garner. Types are weak ω-groupoids. *Proceedings of the London Mathematical Society*, 102(2):370–394, 2011. 23

[76] Vladimir Voevodsky. A very short note on homotopy λ-calculus. *Unpublished*, page 1–7, September 2006. 1, 23

[77] Vladimir Voevodsky. Univalent foundations of mathematics. Invited talk at WoLLIC 2011 18th Workshop on Logic, Language, Information and Computation, 2011. 23

[78] Michael A. Warren. *Homotopy theoretic aspects of constructive type theory*. PhD thesis, Carnegie Mellon University, 2008. 23

[79] Matthew Z. Weaver. A model of type theory with directed univalence in bicubical sets. Talk at the 1st International Conference on Homotopy Type Theory, available from https://hott.github.io/HoTT-2019//conf-slides/Weaver.pdf, 2019. 76, 108

[80] Matthew Z. Weaver. A constructive model of directed univalence in bicubical sets. Talk at the Homotopy Type Theory Electronic Seminar Talks, available from https://www.uwo.ca/math/faculty/kapulkin/seminars/hottestfiles/Weaver-2020-04-16-HoTTEST.pdf, 2020. 76, 108

[81] Matthew Z. Weaver. A constructive model of directed univalence in bicubical sets. Talk at the International Congress on Mathematical Software, 2020. 76, 108

[82] Matthew Z. Weaver. Theory and implementation of bicubical directed type theory. Talk at the 2nd International Conference on Homotopy Type Theory, available from https://www.cs.princeton.edu/~mzweaver/HoTT23/, 2023. 140, 171

[83] Matthew Z. Weaver and Daniel R. Licata. A constructive model of directed univalence in bicubical sets. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 915–928, 2020. 76, 108