CUBICAL METHODS IN HOMOTOPY TYPE THEORY AND UNIVALENT FOUNDATIONS

ANDERS MÖRTBERG

Abstract. Cubical methods have played an important role in the development of Homotopy Type Theory and Univalent Foundations (HoTT/UF) in recent years. The original motivation behind these developments was to give constructive meaning to Voevodsky's univalence axiom, but they have since then led to a range of new results. Among the achievements of these methods is the design of new type theories and proof assistants with native support for notions from HoTT/UF, syntactic and semantic consistency results for HoTT/UF, as well as a variety of independence results and establishing that the univalence axiom does not increase the proof theoretic strength of type theory. This paper is based on lecture notes that were written for the 2019 Homotopy Type Theory Summer School at Carnegie Mellon University. The goal of these lectures was to give an introduction to cubical methods in HoTT/UF and provide sufficient background in order to make the current research in this very active area of HoTT/UF more accessible to newcomers. The focus of these notes is hence on both the syntactic and semantic aspects of these methods, in particular on cubical type theory and the various cubical set categories that give meaning to these theories.

Contents

Acknowledgments: The author is very grateful to Carlo Angiuli and Elisabeth Bonnevier for commenting on earlier versions of these notes. Multiple explanations and diagrams are borrowed from the Ph.D. thesis of Angiuli [Ang19]. The author would also like to thank the organizers of the 2019 Homotopy Type Theory Summer School for inviting him to lecture and the students that attended the school for their many good questions that helped form these notes.

The material in this paper is based upon research supported by the Swedish Research Council (Vetenskapsrådet) under Grant No. 2019-04545.

1. Introduction

This paper is based on lecture notes for a course given at the 2019 Homotopy Type Theory Summer School at Carnegie Mellon University in Pittsburgh.[1] The course covered the basics of cubical type theory with its semantics in cubical sets and this paper closely follows the structure of the course. This means that it is not meant to be a regular research paper with new results, but rather an exposition of cubical type theory and cubical set models. These new type theories provide computational justifications to Homotopy Type Theory and Univalent Foundations (HoTT/UF), in particular, the univalence axiom of Voevodsky [Voe14] is provable and has computational content. This can hence be seen as a constructive foundation for HoTT/UF, suitable for computer implementation.

The univalence axiom of Voevodsky [Voe14] is at the center for HoTT/UF and provides a type theoretic rendering of the idea that all constructions in mathematics should be invariant up to some form of equivalence. This is very common informal practice in mathematics, for instance a mathematician typically makes no distinction between the quotient ring $R/(0)$ and R itself even though they formally are different objects. The univalence axiom makes this practice formal by enabling the identification of these objects in type theory. This way the difference between informal mathematical practice and formalized mathematics can be reduced.

Another interesting aspect of the univalence axiom is that it solves many problems related to the lack of extensionality principles in intensional type theory. For instance, both function extensionality (pointwise equal functions are equal) and propositional extensionality (logically equivalent propositions[2] are equal) are consequences of univalence. Furthermore, one can use it to construct well-behaved quotient types [Uni13; Voe15]. These notions have so far been missing from most type theories, leading to many difficulties in formalizing modern mathematics.

However, while the univalence axiom makes type theory more appealing for formalizing mathematics it also breaks some of the good properties of type theory. In particular, by adding univalence axiomatically one looses canonicity. That is, one can easily construct terms of natural number type in closed context that do not reduce to a numeral. As a consequence, this also breaks existence properties: even if an existence statement is proved using a Σ-type one cannot necessarily extract a witness. This means that

[1]This course was in turn based on a series of lectures given at Inria Sophia Antipolis in May 2017. Those lectures covered the basics of cubical type theory through the `cubicaltt` system and the lecture notes can be found at: https://github.com/mortberg/cubicaltt/tree/master/lectures

[2]By "proposition" we mean "homotopy propositions" or "-1-types". That is, types where all elements are equal.

adding the univalence axiom, at least a priori, breaks the constructive nature of type theory.

This led multiple researchers on a quest to find a constructive justification to the univalence axiom in order to get the extensionality principles of univalence without loosing good type theoretic properties like canonicity. The consistency of the univalence axiom was originally proved by Voevodsky's Kan simplicial set model [KL12]. However, this model used classical logic in crucial ways [BCP15] which led experts to look in different directions for a constructive model. The first major breakthrough was when Bezem et al. [BCH14] formulated the first *cubical* model of HoTT/UF which constituted the starting point of the use of cubical methods in HoTT/UF.

Outline. This paper introduces cubical methods in HoTT/UF by focusing on the cubical type theory developed by Cohen et al. [Coh+18]. This type theory builds on a cubical model of HoTT/UF that has a lot more structure than the original one of Bezem et al. [BCH14]. This additional structure simplifies the model construction and made it possible to formulate a type theory inspired by it. This paper also discusses another class of models based on cartesian cubical sets [AFH18; Ang+17; Awo18] which has led to the development of cartesian cubical type theories. The paper is organized as follows:

- It first introduces both the type theoretical and semantical setting in which the rest of the paper is formulated (Section 2),
- it then continues with a general introduction to cubical type theories and their models (Section 3),
- this is followed by a discussion of the cubical transport operation and why it is not sufficient to get a constructive model of HoTT/UF (Section 4),
- this then naturally leads to the more general cubical Kan composition operations that lets us correct the sides of transported elements (Section 5),
- in order to be able to prove the univalence axiom and give it computational content Glue types are then introduced (Section 6), and
- finally the paper concludes with some suggestions for further reading about cubical methods in HoTT/UF (Section 7).

Furthermore, most sections end with a list exercises that complements the material and which were given to the students during the summer school. Formalized solutions to most exercises can be found in the Cubical Agda library (`https://github.com/agda/cubical/`) and in the `cubicaltt` library (`https://github.com/mortberg/cubicaltt/tree/master/examples`).

2. Background

Homotopy Type Theory and Univalent Foundations, as formulated by Voevodsky [Voe10; Voe11; Voe14; Voe15] and in the HoTT book [Uni13], are axiomatic extensions of type theory as formulated by the Swedish logician Per Martin-Löf. There are multiple different type theories of this kind, see e.g. Martin-Löf [Mar75; Mar82; Mar84; Mar98], and we will here use the term "Martin-Löf type theory" (MLTT) for type theories specified in the MLTT tradition. That is, type theories specified by the four hypothetical judgments:

$$\Gamma \vdash A \qquad \Gamma \vdash A = B \qquad \Gamma \vdash a : A \qquad \Gamma \vdash a = b : A$$

We write $\Gamma \vdash \mathcal{J}$ for an arbitrary judgment and in the rest of the paper we assume that the type theories we work with support Π- and Σ-types with judgmental η rules.[3] We also assume an infinite hierarchy of universes $\mathcal{U}_n$, however we will leave the universe level n implicit and not be specific about the exact rules that these universes satisfy in order to avoid formal technicalities. Basic datatypes like empty, unit, boolean, and natural number types are also assumed and inhabit the lowest universe $\mathcal{U}_0$. We will write functions on these using pattern-matching equations, however all examples can easily be manually translated to recursors and eliminators. We follow the Agda convention (introduced by Nuprl [Con+85]) of writing dependent function types as $(x : A) \to B$ and dependent products as $(x : A) \times B$. The non-dependent variants of these are written $A \to B$ and $A \times B$. We write $p.1$ and $p.2$ for the first and second projections of a pair $p : (x : A) \times B$.

However, we do not include Martin-Löf [Mar75] identity types as in HoTT/UF. We will instead consider *path types* that behave like identity types, but which are not inductively generated. This is achieved by first adding an abstract interval type $\mathbb{I}$ and then defining paths in a type A as the function type $\mathbb{I} \to A$. In this sense cubical type theory is even closer to the homotopical intuition that motivates HoTT/UF–equalities are really *defined* to be paths. The main part of the work will then be to extend the theory so that path types behave like identity types, in particular making the path elimination principle J provable with computational content. The first attempt to solve this is by introducing transport operations, however, it turns out that these are not sufficient as we cannot properly define them for path types themselves. This then leads to the more general Kan composition operations that exist in some form in all cubical type theories and cubical set models.

This, or some variation of it, is essentially the underlying type theoretic setup in the various cubical systems that have been implemented in recent years. These include `cubical` [cubical], `cubicaltt` [cubicaltt], `yacctt` [yacctt], `RedPRL` [Ang+18; Red16], `redtt` [Red18] and Cubical Agda [VMA19]. All of these systems build on different cubical type theories and have different standard cubical models [AFH18; Ang+17; BCH14; BCH18; CH19; CHM18; Coh+18], however the ideas underlying them are very similar and one of the goals of this paper is to give sufficient background to understand and work with the various systems. The two cubical systems that are most actively developed at the time of writing are `redtt` and Cubical Agda. This paper will mainly present the theory underlying Cubical Agda, but many of the ideas and constructions translate directly to the other systems.

The cubical type theories underlying these systems are typically justified using categorical semantics in various cubical set categories. These are often formulated using one of the many frameworks for semantics of MLTT, like the categories with families (CwF's) of Dybjer [Dyb96]. We will be informal in these notes and not commit to a specific framework. However, a crucial result that we will rely on is the fact that any presheaf category forms a CwF (or some other equivalent notion of model) with Π-, Σ-types, and basic datatypes like natural numbers, together with universes closed under these type formers. These results can be found in the lecture notes of [Hof97, Section 4] and the unpublished note about universes of Hofmann and Streicher [HS97].

[3]We say that an equality holds "judgmentally" if it is a judgmental equality specified by the type theory, i.e. if the judgment $\Gamma \vdash a = b : A$ is derivable in the theory. We will refer to the semantic version of this judgment as a "strict" equality, i.e. an equality that holds strictly in some model of the theory.

An elegant way of describing the semantics of cubical type theory and HoTT/UF is to use the internal language of the presheaf topos of cubical sets, following Orton and Pitts [OP18]. As these categories are locally cartesian closed this internal language is extensional type theory, which means that the semantics can be presented using type theoretic notations. In order to avoid confusion we write $\Pi(x : A).B$ and $\Sigma(x : A).B$ for the semantic versions of dependent function and product types. This lets us mimic many of the syntactic constructions from cubical type theory also in the semantics, leading to very short and elegant constructions. However, it is sometimes illuminating or even necessary to express things *externally*, i.e. present the semantics using standard categorical language. We will in the paper make it clear when the semantics is described internally or externally.

3. Cubical type theories and their models

The crucial idea in order to make a type theory *cubical* is to add a primitive interval $\mathbb{I}$ and allow the judgmental structure of the theory to also include contexts with interval variables. Intuitively one may think of $\mathbb{I}$ as a formal analogue of the real interval $[0,1] \subset \mathbb{R}$. A variable $i : \mathbb{I}$ should be thought of as a point that is varying continuously between the two distinct endpoints $0 : \mathbb{I}$ and $1 : \mathbb{I}$:

$$0 \xrightarrow{\quad \overset{i}{\bullet} \quad} 1$$

By extending the judgmental structure of MLTT with these variables we get *cubical* judgments of the form

$$i_1 : \mathbb{I}, ..., i_n : \mathbb{I} \vdash \mathcal{J}$$

Given a judgment $\mathcal{J}$ depending on an interval variable $i : \mathbb{I}$ we write $\mathcal{J}(r/i)$ for $\mathcal{J}$ with r substituted for i. So $A(0/i)$ is a type where 0 has been substituted for i, etc. These substitutions act like regular substitutions, so in particular they behave in the standard way with respect to binders and commute with all type and term formers of the theory. Types and terms in a context with n dimension variables correspond to n-dimensional cubes as described by the following table:

$\vdash A : \mathcal{U}$	$\bullet A$
$i : \mathbb{I} \vdash A : \mathcal{U}$	$A(0/i) \xrightarrow{A} A(1/i)$
$i : \mathbb{I}, j : \mathbb{I} \vdash A : \mathcal{U}$	$\begin{tikzcd} A(0/i)(1/j) \arrow[r, "A(1/j)"] & A(1/i)(1/j) \\ A(0/i)(0/j) \arrow[u, "A(0/i)"] \arrow[r, "A(0/j)"'] \arrow[ur, phantom, "A"] & A(1/i)(0/j) \arrow[u, "A(1/i)"'] \end{tikzcd}$
$\vdots$	$\vdots$

By the standard rules for substitutions we have $A(0/i)(0/j) = A(0/j)(0/i)$, etc. These equations correspond to the lines in the square matching up, namely the source of the left-most line, $A(0/i)$ is the same as the source of the bottom one in the square, $A(0/j)$.

Semantically all cubical set models are based on presheaves on some cube category $\mathcal{C}$, i.e. functor categories $\widehat{\mathcal{C}} := [\mathcal{C}^{\mathrm{op}}, \mathbf{Set}]$. The contexts are modeled by cubical sets and substitutions (or context maps) are modeled by natural transformations between these cubical sets. A very important cube category is the *cartesian* one.

Definition 1. The *cartesian* cube category $\Box$ has as objects finite sets, and as morphisms $\mathrm{Hom}_\Box(I, J)$ functions[4] $J \to I + \{0,1\}$ with identity and composition given by

$$1_I(x) = \mathsf{inl}(x)$$

$$(g \circ f)(x) = \begin{cases} \mathsf{inr}(\varepsilon) & \text{if } g(x) = \mathsf{inr}(\varepsilon) \text{ for } \varepsilon \in \{0,1\} \\ f(y) & \text{if } g(x) = \mathsf{inl}(y) \end{cases}$$

Note that the composition operation is the Kleisli composition for the monad $\_ + \{0,1\}$. We will use the more compact description using the Kleisli category when introducing other cubical set categories later in the paper, but this can always be unfolded into an explicit definition as in the definition of $\Box$ above.

Remark 2. The category $\Box$ is equivalent to the free finite product category on an interval object [Awo18; Par14].

We write $I, J, K, ...$ for objects of $\Box$, and say that a finite set $\{i_1, \dots, i_n\}$ is an n-cube of *dimensions* $i_1, \dots, i_n$. Notable morphisms in the category $\Box$ include:

- Given $\varepsilon \in \{0,1\}$, a dimension i and a finite set I there are *face maps*
$$d^i_\varepsilon \in \mathrm{Hom}_\Box(I, I + \{i\})$$
$$d^i_\varepsilon(j) = \begin{cases} \varepsilon & \text{if } i = j \\ j & \text{otherwise} \end{cases}$$
- Given a dimension i and a finite set I there are *degeneracy maps*
$$s^i \in \mathrm{Hom}_\Box(I + \{i\}, I)$$
$$s^i(j) = j$$
- Given dimensions i, j and a finite set I there are *symmetry maps*
$$t^{i,j} \in \mathrm{Hom}_\Box(I + \{j,i\}, I + \{i,j\})$$
$$t^{i,j}(k) = \begin{cases} j & \text{if } k = i \\ i & \text{if } k = j \\ k & \text{otherwise} \end{cases}$$
- Given dimensions i, j and a finite set I there are *diagonal maps*
$$c^{i,j} \in \mathrm{Hom}_\Box(I + \{i\}, I + \{i,j\})$$
$$c^{i,j}(k) = \begin{cases} i & \text{if } k = i \text{ or } k = j \\ k & \text{otherwise} \end{cases}$$

[4]The reason I and J are flipped in the functions is that we will take the opposite of this category when defining cartesian cubical sets.

A cartesian cubical set is a functor $\Gamma : \widehat{\square}$. Geometrically we may think of such a Γ as a space and $\Gamma(\{i_1, \dots, i_n\})$ as the set of continuous functions $[0,1]^n \to \Gamma$. The face maps $\Gamma(d^i_\varepsilon) : \Gamma(I + \{i\}) \to \Gamma(I)$ restrict $(n+1)$-cubes to n-cubes by setting the i coordinate to ε. The degeneracy maps $\Gamma(s^i) : \Gamma(I) \to \Gamma(I + \{i\})$ lets us regard n-cubes as $(n+1)$-cubes. The symmetry maps rotate cubes by permuting the axes. Finally, the diagonal maps extract the various diagonal n-cubes of $(n+1)$-cubes. We illustrate the actions of these maps informally below for various concrete n-cubes:

Faces
$$\begin{tikzcd} \bullet \arrow[r, "s"] & \bullet \\ \bullet \arrow[u, "r"] \arrow[r, "p"'] & \bullet \arrow[u, "q"'] \end{tikzcd} \quad \mapsto \quad \bullet \xrightarrow{p} \bullet \quad \bullet \xrightarrow{q} \bullet \quad \bullet \xrightarrow{r} \bullet \quad \bullet \xrightarrow{s} \bullet$$

Degeneracies
$$a \xrightarrow{p} b \quad \mapsto \quad \begin{tikzcd} a \arrow[r, "p"] \arrow[d, equal] & b \arrow[d, equal] \\ a \arrow[r, "p"'] & b \end{tikzcd}$$

Symmetries
$$\begin{tikzcd} \bullet \arrow[r, "s"] & \bullet \\ \bullet \arrow[u, "r"] \arrow[r, "p"'] & \bullet \arrow[u, "q"'] \end{tikzcd} \quad \mapsto \quad \begin{tikzcd} \bullet \arrow[r, "q"] & \bullet \\ \bullet \arrow[u, "p"] \arrow[r, "r"'] & \bullet \arrow[u, "s"'] \end{tikzcd}$$

Diagonals
$$\begin{tikzcd} \bullet \arrow[r] & b \\ a \arrow[u] \arrow[r] & \bullet \arrow[u] \end{tikzcd} \quad \mapsto \quad a \longrightarrow b$$

These satisfy various evident cubical identities, e.g. degenerating and taking the corresponding face does nothing. We will now see that there is a close relationship between these identities and the rules satisfied by the cubical judgments of cubical type theory.

Let $\mathbf{y} : \square \to \widehat{\square}$ be the Yoneda embedding defined at an object $I \in \square$ as $\mathbf{y}(I) := \mathrm{Hom}_\square(\_, I)$. The interval of cubical type theory is modeled by $\mathbf{y}(\{i\})$ where i is an arbitrary dimension. We write $\mathbb{I}$ for this representable 1-cube as well. An important property of cubical sets is that the product of representable cubical sets is again representable.[5] Combined with the fact that $\mathbf{y}$ preserves products we get that the representable n-cube is an n-fold product of $\mathbb{I}$, i.e. $\mathbf{y}(\{i_1, \dots, i_n\}) \simeq \mathbb{I}^n$. The Yoneda lemma hence gives a bijection between the n-cubes of a cubical set Γ and natural transformations $\mathbb{I}^n \to \Gamma$. This means that the structure of a cubical set is determined by maps

[5]This is not true for simplicial sets where the product of representables has to be subdivided in order to form a simplicial set again.

out of products of cubical sets, justifying the geometric intuition above. Furthermore, a type $i_1 : \mathbb{I}, \ldots, i_n : \mathbb{I} \vdash A : \mathcal{U}$ corresponds to a morphism $A : \mathbb{I}^n \to \mathcal{U}$, justifying the cubical judgments. This is one reason why cubical sets are so well-suited as a basis for higher dimensional type theory.

Given an n-cube $I = \{i_1, \ldots, i_n\}$ and context $\Gamma = i_1 : \mathbb{I}, \ldots, i_n : \mathbb{I}$ we can now clarify the relationship between the proof theory of cubical type theory and its semantics as in the table below.

Syntax/proof theory	Semantics
$\dfrac{\Gamma, i : \mathbb{I} \vdash \mathcal{J}}{\Gamma \vdash \mathcal{J}(\varepsilon/i)}$ face	$\mathbf{y}(I) \xrightarrow{\mathbf{y}(d^i_\varepsilon)} \mathbf{y}(I + \{i\})$
$\dfrac{\Gamma \vdash \mathcal{J}}{\Gamma, i : \mathbb{I} \vdash \mathcal{J}}$ weakening	$\mathbf{y}(I + \{i\}) \xrightarrow{\mathbf{y}(s^i)} \mathbf{y}(I)$
$\dfrac{\Gamma, i : \mathbb{I}, j : \mathbb{I} \vdash \mathcal{J}}{\Gamma, j : \mathbb{I}, i : \mathbb{I} \vdash \mathcal{J}}$ exchange	$\mathbf{y}(I + \{j, i\}) \xrightarrow{\mathbf{y}(t^{i,j})} \mathbf{y}(I + \{i, j\})$
$\dfrac{\Gamma, i : \mathbb{I}, j : \mathbb{I} \vdash \mathcal{J}}{\Gamma, i : \mathbb{I} \vdash \mathcal{J}(i/j)}$ contraction	$\mathbf{y}(I + \{i, j\}) \xrightarrow{\mathbf{y}(c^{i,j})} \mathbf{y}(I + \{i\})$

This shows that there is a close correspondence between the maps in the cubical set categories and the structure of the cubical judgments in cubical type theory. This correspondence was analyzed by Buchholtz and Morehouse [BM17] to define and relate a great variety of cubical set categories.

Remark 3. If we omit diagonals we obtain the *substructural* cubical sets that underlie the original cubical set model of Bezem et al. [BCH14]. However, because of the substructural nature of this cube category it is not as easy to develop a cubical type theory based on this model (even for non-cubical type theories substructural dependent type theory is an active research area).

3.1. **Path types.** In order to be able to talk about paths between terms and between types in cubical type theory we need to extend the theory with path types. These types are a type theoretic rendering of the idea that a path is just a function out of the interval in cubical type theory. The rules for these follow below.

$$\frac{\Gamma, i : \mathbb{I} \vdash A \qquad \Gamma \vdash a : A(0/i) \qquad \Gamma \vdash b : A(1/i)}{\Gamma \vdash \mathsf{Path}^i\ A\ a\ b}$$

$$\frac{\Gamma, i : \mathbb{I} \vdash A \qquad \Gamma, i : \mathbb{I} \vdash a : A}{\Gamma \vdash \lambda(i : \mathbb{I}).\ a : \mathsf{Path}^i\ A\ a(0/i)\ a(1/i)} \qquad \frac{\Gamma \vdash p : \mathsf{Path}^i\ A\ a\ b \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash p\ r : A(r/i)}$$

$$\frac{\Gamma, i : \mathbb{I} \vdash A \qquad \Gamma, i : \mathbb{I} \vdash a : A \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash (\lambda(i : \mathbb{I}).\ a)\ r = a(r/i) : A(r/i)}\beta \qquad \frac{\Gamma \vdash p : \mathsf{Path}^i\ A\ a\ b}{\Gamma \vdash (\lambda(j : \mathbb{I}).\ p\ j) = p : \mathsf{Path}^i\ A\ a\ b}\eta$$

$$\frac{\Gamma \vdash p : \mathsf{Path}^i\ A\ a\ b}{\Gamma \vdash p\ 0 = a : A(0/i)} \qquad \frac{\Gamma \vdash p : \mathsf{Path}^i\ A\ a\ b}{\Gamma \vdash p\ 1 = b : A(1/i)}$$

Note that these rules are very similar to those of Π-types, except that special care has to be taken when applying a path to one of the endpoints of the interval. If A doesn't depend on i we write simply $\mathsf{Path}\ A\ a\ b$. These non-dependent path types are supposed to correspond to the identity types of HoTT/UF while Path^i is a cubical version of the "path-over" types of HoTT, i.e. paths living over a line of types. Having built-in path-over types is very useful, especially when working with higher inductive types, as we can directly represent heterogeneous equalities (i.e. equalities between terms of different types). Furthermore, representing equalities using path types allows direct definitions of many standard operations on identity types that are usually proved by identity elimination.

For example, given $A : \mathcal{U}$ we define a proof of reflexivity as a constant path (i.e. a constant function):

$\mathsf{refl} : (a : A) \to \mathsf{Path}\ A\ a\ a$

$\mathsf{refl}\ a = \lambda(i : \mathbb{I}).\ a$

Remark 4. Note that, as opposed to identity types, the path types are not inductively generated. This means that we have to be able to prove both refl and J (path induction).

Given $A, B : \mathcal{U}$ we can also prove that the images of two path-equal elements are path-equal

$\mathsf{ap} : (a\ b : A)\ (f : A \to B)\ (p : \mathsf{Path}\ A\ a\ b) \to \mathsf{Path}\ B\ (f\ a)\ (f\ b)$

$\mathsf{ap}\ a\ b\ f\ p = \lambda(i : \mathbb{I}).\ f\ (p\ i)$

This operation satisfies some judgmental equalities that do not hold judgmentally when ap is defined using identity elimination on p, for example:

$$\mathsf{ap}\ \mathrm{id}\ p = p$$
$$\mathsf{ap}\ (g \circ f)\ p = \mathsf{ap}\ f\ (\mathsf{ap}\ g\ p)$$

The fact that we get these equations judgmentally is very useful when formalizing mathematics. We can also define new operations that doesn't always hold for identity types, for instance, function extensionality for path types can be proved as:

$\mathsf{funext} : (f\ g : (x : A) \to B)\ (p : (x : A) \to \mathsf{Path}\ B\ (f\ x)\ (g\ x)) \to \mathsf{Path}\ ((x : A) \to B)\ f\ g$

$\mathsf{funext}\ f\ g\ p = \lambda(i : \mathbb{I}).\ \lambda(x : A).\ p\ x\ i$

To see that this has the correct boundaries we do the following computation:

$$(\lambda(i : \mathbb{I}).\ \lambda(x : A).\ p\ x\ i)\ 0 = \lambda(x : A).\ p\ x\ 0 = \lambda(x : A).\ f\ x = f$$

Note that the last equality holds by the η rule for Π-types. The case for the right endpoint (i.e. when applying funext to 1) holds by an analogous computation.

Equality in Σ-types is notoriously complicated to work with in MLTT, but with dependent path types things get more manageable. Given $A : \mathcal{U}$ and a family $B : A \to \mathcal{U}$ we define

$$\begin{aligned}&\Sigma_{\mathsf{eq}} : (s\ t : (x : A) \times B)\ (p : \mathsf{Path}\ A\ s.1\ t.1)\ (q : \mathsf{Path}^i\ (B\ (p\ i))\ s.2\ t.2) \to \\ &\quad \mathsf{Path}\ ((x : A) \times B)\ s\ t \\ &\Sigma_{\mathsf{eq}}\ s\ t\ p\ q = \lambda(i : \mathbb{I}).\ (p\ i, q\ i)\end{aligned}$$

Working with path-over types like this is very convenient as no transports are necessary, making reasoning with equalities in Σ-types a lot easier than in traditional MLTT.

Semantically we can justify path types using the internal language of $\widehat{\square}$. Given a family of types $A : \mathbb{I} \to \mathcal{U}$ we define:

$$\mathsf{Path}(A) := \Pi(i : \mathbb{I}).A(i)$$

We then define the type of paths between $a : A(0)$ and $b : A(1)$ as

$$\mathsf{Path}_A(a, b) := \Sigma(p : \mathsf{Path}(A)).(p\ 0 = a \wedge p\ 1 = b)$$

It is easy to verify that this satisfies the rules of path types as they are constructed using semantic Π- and Σ-types. Furthermore, the exact same operations as above are easily definable semantically.

3.2. **Connections and reversals.** It is often very useful to assume more structure on the underlying cube category, both when constructing models and for making proofs simpler in the cubical type theory based on the model. This is the done in both `cubicaltt` and Cubical Agda which are based on the cube category of the CCHM cubical set model of Cohen et al. [Coh+18]. Before defining this category we have to introduce De Morgan algebras:

Definition 5. A bounded distributive lattice $(A, 0, 1, \wedge, \vee)$ is a De Morgan algebra if it has an involution $\neg : A \to A$ satisfying the De Morgan identities:

$$\neg(r \vee s) = \neg r \wedge \neg s \qquad\qquad \neg(r \wedge s) = \neg r \vee \neg s$$

We write DM for the monad on the category of sets associating to each set A the free De Morgan algebra on A.

Definition 6. The *De Morgan* cube category $\square_{\mathsf{DM}}$ has as objects finite sets, and as morphisms $\mathrm{Hom}_{\square_{\mathsf{DM}}}(I, J)$ functions $J \to \mathsf{DM}(I)$. Identity and composition inherited from the Kleisli category of DM.

This category has all the morphisms of $\square$, but there are very many more (see exercise 8). Notable new morphisms in this category are:

- Given a dimension i and a finite set I there are *connection maps*

$$\begin{aligned}&c^i_\wedge \in \mathrm{Hom}_{\square_{\mathsf{DM}}}(I + \{i\}, I) &\qquad& c^i_\vee : \mathrm{Hom}_{\square_{\mathsf{DM}}}(I + \{i\}, I) \\ &c^i_\wedge(j) = i \wedge j && c^i_\vee(j) = i \vee j\end{aligned}$$

- Given a dimension i and a finite set I there are *reversal* maps:

$$r^i \in \mathrm{Hom}_{\square_{\mathsf{DM}}}(I + \{i\}, I)$$

$$r^i(j) = \begin{cases} \neg i & \text{if } i = j \\ j & \text{otherwise} \end{cases}$$

A CCHM cubical set is a functor $\Gamma : \widehat{\square}_{\mathsf{DM}}$. The connections can be thought of as new kinds of degeneracies while the reversals lets us invert lines as illustrated in the table below.

Connections $a \xrightarrow{p} b \quad \mapsto$

$$\begin{tikzcd} b \arrow[r, equal] & b \\ a \arrow[u, "p"] \arrow[r, "p"'] \arrow[ur, phantom, "\vee"] & b \arrow[u, equal] \end{tikzcd} \qquad \begin{tikzcd} a \arrow[r, "p"] & b \\ a \arrow[u, equal] \arrow[r, equal] \arrow[ur, phantom, "\wedge"] & a \arrow[u, "p"'] \end{tikzcd}$$

Reversals $a \longrightarrow b \quad \mapsto \quad b \longrightarrow a$

The interval in $\widehat{\square}_{\mathsf{DM}}$ is defined in the same way as the one in $\widehat{\square}$ using the Yoneda embedding. We may see the connection and reversal morphisms in $\widehat{\square}_{\mathsf{DM}}$ as operations in $\widehat{\square}_{\mathsf{DM}}$ of type $\wedge, \vee : \mathbb{I} \to \mathbb{I} \to \mathbb{I}$ and $\neg : \mathbb{I} \to \mathbb{I}$ satisfying the axioms of a De Morgan algebra. The topological intuition behind these operations is that $r \wedge s$ corresponds to $\min(r, s)$, $r \vee s$ to $\max(r, s)$ and $\neg r$ to $1 - r$ for $r, s \in [0, 1]$.

Remark 7. One might wonder why De Morgan algebras and not for example Boolean algebras? The reason is that Boolean algebras do not describe the theory of the real interval. Indeed, the equations $r \wedge \neg r = 0$ and $r \vee \neg r = 1$ are not generally true for $r \in [0, 1]$ except for at the endpoints. However, despite the interval in Boolean algebras not satisfying the same axioms as the real interval there is no problem with constructing a model using it.

Type theoretically we may now substitute interval variables for formulas built using connections and reversals in order to construct more complex cubes out of simpler ones. For example, given $p : \mathsf{Path}\ A\ a\ b$ and $i, j : \mathbb{I}$ we can construct the connection squares $p\ (i \wedge j)$ and $p\ (i \vee j)$ as

$$\begin{tikzcd} a \arrow[r, "p\ i"] & b \\ a \arrow[u, "p\ 0"] \arrow[r, "p\ 0"'] \arrow[ur, phantom, "p\ (i \wedge j)"] & a \arrow[u, "p\ j"'] \end{tikzcd} \qquad \begin{tikzcd} b \arrow[r, "p\ 1"] & b \\ a \arrow[u, "p\ j"] \arrow[r, "p\ i"'] \arrow[ur, phantom, "p\ (i \vee j)"] & b \arrow[u, "p\ 1"'] \end{tikzcd} \qquad \begin{tikzcd} {} \\ \cdot \arrow[u, "j"] \arrow[r, "i"'] & {} \end{tikzcd}$$

where, for instance, the right-hand side of the left square is computed as

$$p\ (i \wedge j)(1/i) = p\ (1 \wedge j) = p\ j$$

The ability to directly construct squares with boundaries given by the formulas that can be formed in a De Morgan algebra is very convenient. An example of this is the proof that singletons are contractible, that is: any element in $(x : A) \times (\mathsf{Path}\ A\ a\ x)$ is path-equal to (a, refl_a).

$$\mathsf{contrSingl} : (a\ b : A)\ (p : \mathsf{Path}\ A\ a\ b) \to \mathsf{Path}\ ((x : A) \times (\mathsf{Path}\ A\ a\ x))\ (a, \mathsf{refl}_a)\ (b, p)$$
$$\mathsf{contrSingl}\ a\ b\ p = \lambda(i : \mathbb{I}).\ (p\ i, \lambda(j : \mathbb{I}).\ p\ (i \wedge j))$$

Obviously the first component is a path between a and b. The second component is a path from $\lambda(j : \mathbb{I}).\ p\ 0$ to $\lambda(j : \mathbb{I}).\ p\ j$, i.e. from refl_a to p by the computation and η rules for path types.

Given $a, b : A$ we may also define the symmetry of a path as follows:

$$\mathsf{sym} : \mathsf{Path}\ A\ a\ b \to \mathsf{Path}\ A\ b\ a$$
$$\mathsf{sym}\ p = \lambda(i : \mathbb{I}).\ p\ (\neg i)$$

This satisfies $\mathsf{sym}\ (\mathsf{sym}\ p) = p$ judgmentally. This is another example of an equation that doesn't hold judgmentally when sym is defined by induction on p and is useful when formalizing mathematics, for example we may directly define the opposite of a category so that $\mathcal{C}^{\mathrm{op}^{\mathrm{op}}} = \mathcal{C}$ judgmentally.

With this new additional structure the path types almost behave like the identity types of HoTT/UF, however we need to add additional structure that lets us prove the path-elimination principle commonly referred to as J following Martin-Löf [Mar75]:

$$\mathsf{J} : (A : \mathcal{U})\ (a : A)\ (C : (x : A) \to \mathsf{Path}\ A\ a\ x \to \mathcal{U})$$
$$(d : C\ a\ \mathsf{refl}_a)\ (x : A)\ (p : \mathsf{Path}\ A\ a\ x) \to C\ x\ p$$

The J operation may in fact be decomposed as contractibility of singletons (which we have already proved) and subst (see exercise (10)):

$$\mathsf{subst} : (A : \mathcal{U})\ (P : A \to \mathcal{U})\ (a\ b : A)\ (p : \mathsf{Path}\ A\ a\ b)\ (e : P\ a) \to P\ b$$

In fact, in the next section we will reduce subst to an even simpler operation that we call $\mathsf{transport}$.[6] We will then equip cubical sets with a structure corresponding to this operation and explain how it computes in the type theory and why the semantic type formers are closed under this structure.

3.3. Exercises.

(1) Prove that $\{i\} \times \{j\} \simeq \{i, j\}$ in $\square$. More generally, prove that $\square$ has finite products.

(2) Prove the binary version of ap

$$\mathsf{ap}_2 : (a\ a' : A)\ (b\ b' : B)\ (f : A \to B \to C)$$
$$(p : \mathsf{Path}\ A\ a\ a')\ (q : \mathsf{Path}\ B\ b\ b') \to \mathsf{Path}\ C\ (f\ a\ b)\ (f\ a'\ b')$$

(3) Prove the binary (non-dependent) version of function extensionality

$$\mathsf{funext}_2 : (f\ g : A \to B \to C)\ (p : (x : A)\ (y : B) \to \mathsf{Path}\ C\ (f\ x\ y)\ (g\ x\ y)) \to$$
$$\mathsf{Path}\ (A \to B \to C)\ f\ g$$

(4) Define negation on booleans $\mathsf{not} : \mathsf{bool} \to \mathsf{bool}$ and prove that

$$\mathsf{notK} : \mathsf{Path}\ (\mathsf{bool} \to \mathsf{bool})\ (\mathsf{not} \circ \mathsf{not})\ \mathsf{id}$$

[6]What we here call subst is often called $\mathsf{transport}$ in HoTT/UF, however as we will reduce subst to a more elementary operation in the next section we call the basic operation $\mathsf{transport}$.

(5) We can define a predicate $\mathsf{isProp} : \mathcal{U} \to \mathcal{U}$ that expresses that a type is an h-proposition as

$$\mathsf{isProp}\ A = (x\ y : A) \to \mathsf{Path}\ A\ x\ y$$

Prove that a family of h-propositions is an h-proposition:

$$\begin{aligned}\mathsf{propPi} : (A : \mathcal{U})\ (B : A \to \mathcal{U})\ (h : (x : A) \to \mathsf{isProp}\ (B\ x)) \to \\ \mathsf{isProp}\ ((x : A) \to B\ x)\end{aligned}$$

(6) Given $p : \mathsf{Path}\ A\ a\ b$ and $i, j : \mathbb{I}$, draw the squares corresponding to
   (a) $p\ (\neg i \wedge j)$
   (b) $p\ (i \wedge \neg j)$
   (c) $p\ (\neg i \wedge \neg j)$
   (d) $p\ (\neg i \vee j)$
   (e) $p\ (i \vee \neg j)$
   (f) $p\ (\neg i \vee \neg j)$

(7) Given $p : \mathsf{Path}\ A\ a\ b$ and $i, j, k : \mathbb{I}$, draw the cubes corresponding to
   (a) $p\ (i \wedge j \wedge k)$
   (b) $p\ (i \wedge \neg j \vee k)$
   (c) $p\ (\neg i \vee \neg j \vee \neg k)$

(8) How many elements does $\mathrm{Hom}_{\square}(\{i_1, \ldots, i_n\}, \{i\})$ have? What about $\mathrm{Hom}_{\square_{\mathsf{DM}}}(\{i_1, \ldots, i_n\}, \{i\})$?

(9) Prove the following variation of $\mathsf{contrSingl}$

$$\begin{aligned}\mathsf{contrSingl}' : (a\ b : A)\ (p : \mathsf{Path}\ A\ a\ b) \to \\ \mathsf{Path}\ ((x : A) \times (\mathsf{Path}\ A\ x\ b))\ (b, \mathsf{refl}_b)\ (a, p)\end{aligned}$$

(10) Prove J using subst and $\mathsf{contrSingl}$. What does $\mathsf{J}\ A\ a\ C\ d\ a\ \mathsf{refl}_a$ evaluate to?

(11) The circle $\mathbb{S}^1$ has constructors base and $\mathsf{loop} : \mathbb{I} \to \mathbb{S}^1$ such that $\mathsf{loop}\ 0 = \mathsf{loop}\ 1 = \mathsf{base}$. Why is this type non-trivial even though $\lambda(i\ j : \mathbb{I}).\ \mathsf{loop}\ (i \wedge j)$ proves that $\mathsf{refl}_{\mathsf{base}}$ is path-equal to loop? (hint: draw a picture)

(12) Construct a homotopy on the circle that shows that the path going halfway around and back is contractible:[7]

$$\mathsf{hmtpy} : \mathsf{Path}\ (\mathsf{Path}\ \mathbb{S}^1\ \mathsf{base}\ \mathsf{base})\ \mathsf{refl}_{\mathsf{base}}\ (\lambda(i : \mathbb{I}).\ \mathsf{loop}\ (i \wedge \neg i))$$

(13) Prove that it is inconsistent to assume decidable equality of $\mathbb{I}$ internally. (hint: construct a path from the unit type to the empty type using decidable equality on $\mathbb{I}$)

4. Transport

In order to be able to prove J we will introduce a new operation that we call *transport*. Type theoretically it can be described by the rule:

$$\frac{\Gamma, i : \mathbb{I} \vdash A : \mathcal{U} \qquad \Gamma \vdash a : A(0/i)}{\Gamma \vdash \mathsf{transport}^i\ A\ a : A(1/i)}$$

Note that $\mathsf{transport}^i$ binds i in A. An alternative definition (taken by Cubical Agda for instance) is to require that A is a path-abstraction. There is no deep reason for doing this either way, except that when working informally on paper the author finds it

[7]Remember that $i \wedge \neg i$ is not necessarily 0 in a De Morgan algebra.

easier to have transport act as a binder (but other experts disagree with this). However, when implementing a proof assistant based on this theory it is typically easier to have as few binders as possible.

With transport we can prove subst as

$$\mathsf{subst} : (A : \mathcal{U})\ (P : A \to \mathcal{U})\ (a\ b : A)\ (p : \mathsf{Path}\ A\ a\ b)\ (e : P\ a) \to P\ b$$

$$\mathsf{subst}\ A\ P\ a\ b\ p\ e = \mathsf{transport}^i\ (P\ (p\ i))\ e$$

One way to intuitively understand this operation is as an operation for transporting a point from one endpoint to the other over a line of types:

$$\begin{tikzcd}
a \bullet & \bullet\ \mathsf{transport}^i\ A\ a \\
A(0/i) \arrow[r, "A"] & A(1/i)
\end{tikzcd}$$

In order to give this operation computational meaning we have to add computation rules for it. This is done by adding different judgmental equalities for the different type formers of the theory. For instance, for non-dependent pairs $A \times B$ we define:

$$\frac{\Gamma, i : \mathbb{I} \vdash A \times B : \mathcal{U} \qquad \Gamma \vdash p : A(0/i) \times B(0/i)}{\Gamma \vdash \mathsf{transport}^i\ (A \times B)\ p = (\mathsf{transport}^i\ A\ p.1, \mathsf{transport}^i\ B\ p.2) : A(1/i) \times B(1/i)}$$

It is easy to see that this is a well-defined equality by checking that the right hand side has the correct type.

For function types $A \to B$ we need to be able to transport *backwards* as well. That is, given $a : A(1/i)$ we want an element in $\mathsf{transport}^{\neg i}\ A\ a : A(0/i)$:

$$\begin{tikzcd}
\mathsf{transport}^{\neg i}\ A\ a \bullet & \bullet\ a \\
A(0/i) \arrow[r, "A"] & A(1/i)
\end{tikzcd}$$

Using a reversal we can define this as:[8]

$$\mathsf{transport}^{\neg i}\ A\ a := \mathsf{transport}^i\ A(\neg i/i)\ a$$

We can now give the definition of transport for $A \to B$: given $\Gamma, i : \mathbb{I} \vdash A \to B : \mathcal{U}$ and $\Gamma \vdash f : A(0/i) \to B(0/i)$ we are going to define

$$\mathsf{transport}^i\ (A \to B)\ f : A(1/i) \to B(1/i)$$

To do this we first abstract over $x : A(1/i)$ and transport it backwards to get an element $\mathsf{transport}^{\neg i}\ A\ x$ of type $A(0/i)$. We then apply f and obtain an element in $B(0/i)$ which can be transported forward in B to get the desired element of $B(1/i)$. We can summarize this construction in the computation rule:

$$\frac{\Gamma, i : \mathbb{I} \vdash A \to B : \mathcal{U} \qquad \Gamma \vdash f : A(0/i) \to B(0/i)}{\Gamma \vdash \mathsf{transport}^i\ (A \to B)\ f = \lambda(x : A(1/i)).\ \mathsf{transport}^i\ B\ (f\ (\mathsf{transport}^{\neg i}\ A\ x)) : A(1/i) \to B(1/i)}$$

[8]Note that we distinguish between *definitions* written using := and judgmental equalities. The definitions are not added as new rules to the theory, but rather as definitions that can be unfolded.

For basic datatypes without parameters we can let transport be the identity function, e.g. for natural numbers we have $\mathsf{transport}^i\ \mathbb{N}\ n = n$. This makes sense as $\mathbb{N}(r/i) = \mathbb{N}$ for any $r : \mathbb{I}$. For parameterized datatypes we can define the transport operation for each constructor (see exercise (2)).

Remark 8. Note that there is a choice in how $\mathsf{transport}$ behaves for pair and function types. We could instead have treated these transports as neutral values that don't reduce further unless we either apply a projection or apply them to something. We call these the *negative definitions* of $\mathsf{transport}$ as the computation rules are defined using eliminators, while the ones we gave above are the *positive definitions*.

We now need to define $\mathsf{transport}$ for Path types. Given $\Gamma, i : \mathbb{I} \vdash \mathsf{Path}\ A\ a\ b$ and $p : (\mathsf{Path}\ A\ a\ b)(0/i)$ we are going to construct an element of $(\mathsf{Path}\ A\ a\ b)(1/i)$. Consider the following naive definition (where j is a fresh dimension variable):

$$\mathsf{transport}^i\ (\mathsf{Path}\ A\ a\ b)\ p = \lambda(j : \mathbb{I}).\ \mathsf{transport}^i\ A\ (p\ j)$$

This might look like a plausible definition, but the resulting path does not have the right endpoints! Indeed, when j is 0 this is $\mathsf{transport}^i\ A\ a$ and not $a(1/i)$. We run into similar problems when trying to naively define transport for Π- and Σ-types (see exercises (3)). The way we solve this, following Cohen et al. [Coh+18], is to generalize transport and instead consider more general *composition* operations.

4.1. **Exercises.**

(1) Give the *negative* definition of transport for $A \times B$.
(2) Define transport in sum types $A + B$. (hint: pattern-match on the constructors)

(3) Try to define $\mathsf{transport}$ for Σ- and Π-types and see what problems you run into.

5. Kan composition operations

In order to solve the problem with transport for path types from the previous section we introduce a generalized transport operation that also lets us fix the boundary of the transported element. Given $p : (\mathsf{Path}\ A\ a\ b)(0/i)$ we write this as:

$$\lambda(j : \mathbb{I}).\ \mathsf{comp}^i\ A\ [(j = 0) \mapsto a, (j = 1) \mapsto b]\ (p\ j)$$

This is an element of $A(1/i)$ where the boundary has been fixed to be a and b. The syntax of the comp operation is the same as the one for $\mathsf{transport}$, except that it also takes a list of faces (written in general as $[(j_0 = \varepsilon_0) \mapsto a_0, \ldots, (j_n = \varepsilon_n) \mapsto a_n]$) that has to match up with $p\ j$. In the above example the list of faces is $[(j = 0) \mapsto a, (j = 1) \mapsto b]$ and these match up with $p\ j$ as p is a path between a and b so that $(p\ j)(0/j) = p\ 0 = a$ and $(p\ j)(1/j) = p\ 1 = b$.

We refer to this operation as *Kan composition* and, as explained above, it is a form of transport where we can fix the sides of the transported element. We can illustrate the above example diagrammatically as follows:

$$\begin{tikzcd}
a \arrow[r, dashed] & b & & {} \\
a \arrow[u, "a"] \arrow[r, "p\ j"'] & b \arrow[u, "b"'] & & {} \arrow[u, "i"] \arrow[r, "j"'] & {}
\end{tikzcd}$$

The vertical sides in the drawing are going in direction i while the bottom (or “base”) is going in direction j. Furthermore, for this operation to be well-formed the sides (or “tubes”) has to match the base, up to judgmental equality, as in the picture.

Another important property of this operation is that when we perform a substitution that makes one of the faces “true” (i.e. turn it into $(0 = 0)$ or $(1 = 1)$) the operation reduces to the corresponding element at this face at 1, so if we substitute 0 for j in the example the composition term turns into $a(1/i) = a$. Furthermore, “false” faces, $(0 = 1)$ and $(1 = 0)$, can be disregarded/deleted. Finally, note that this operations binds i in all of A, a and b, but not in $p\ j$.

Let us now look at some examples of compositions. Given $a, b, c : A$ we can define the concatenation of two paths as:

$$\mathsf{compPath} : \ (p : \mathsf{Path}\ A\ a\ b)\ (q : \mathsf{Path}\ A\ b\ c) \to \mathsf{Path}\ A\ a\ c$$
$$\mathsf{compPath}\ p\ q = \lambda(j : \mathbb{I}).\ \mathsf{comp}^i\ A\ [(j = 0) \mapsto a, (j = 1) \mapsto q\ i]\ (p\ j)$$

This can be illustrated as the dashed line in:

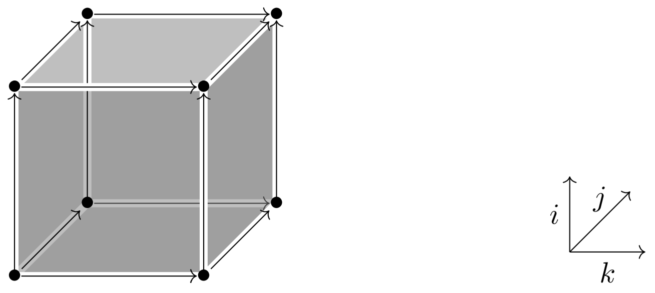
$$\begin{tikzcd}
a \arrow[r, dashed] & c & & {} \\
a \arrow[u, "a"] \arrow[r, "p\ j"'] & b \arrow[u, "q\ i"'] & & {} \arrow[u, "i"] \arrow[r, "j"'] & {}
\end{tikzcd}$$

Let us now prove one of the groupoid laws: the concatenation of a path with its inverse is equal to refl. Given $a, b : A$ we give the following complicated definition:

$$\mathsf{compSym} : (p : \mathsf{Path}\ A\ a\ b) \to \mathsf{Path}\ (\mathsf{Path}\ A\ a\ a)\ (\mathsf{compPath}\ p\ (\mathsf{sym}\ p))\ \mathsf{refl}_a$$
$$\mathsf{compSym}\ p = \lambda(k\ j : \mathbb{I}).\ \mathsf{comp}^i\ A\ [(j = 0) \mapsto a, (j = 1) \mapsto p\ (\neg i \wedge \neg k), (k = 1) \to a]\ (p\ (j \wedge \neg k))$$

In order to see that this is a path between the two desired endpoints we first set k to 0 and then to 1 and see what the term evaluates to. In the first case we obtain $\lambda(j : \mathbb{I}).\ \mathsf{comp}^i\ A[(j = 0) \mapsto a, (j = 1) \mapsto p\ (\neg i)]\ (p\ j)$ after deleting false faces. Up to renaming of bound variables this is the same as $\mathsf{compPath}\ p\ (\mathsf{sym}\ p)$ as desired. In the second case the third face becomes true (i.e. when k is 1 it is $(1 = 1)$) so the whole term reduces to $\lambda(j : \mathbb{I}).\ a$ which is the same as refl_a as desired.

Terms like this might look very daunting at first, but in order to construct them we draw pictures. As we are constructing a path of paths, i.e. a square, we will have to form a 3-dimensional composition drawing, i.e. a cube. We draw this as follows:



The top face is the desired result, i.e a square in direction k with left-hand side being compPath p (sym p) and all three other sides refl_a. This fixes the left ($k = 0$) side to:

$$\begin{tikzcd}
a \arrow[r, dashed] & a & & \\
a \arrow[u, "a"] \arrow[r, "p\ j"'] & b \arrow[u, "p\ (\neg i)"'] & & \bullet \arrow[u, "i"] \arrow[r, "j"']
\end{tikzcd}$$

For the right ($j = 0$) and front ($k = 1$) squares can take a degenerate a (so they are a everywhere). We are then left to find suitable squares for the back ($j = 1$) and bottom ($i = 0$) sides. If we let the bottom be

$$\begin{tikzcd}
b \arrow[r, "p\ (\neg k)"] & a & & \\
a \arrow[u, "p\ j"] \arrow[r, "a"'] \arrow[ur, phantom, "p\ (j \wedge \neg k)"] & a \arrow[u, "a"'] & & \bullet \arrow[u, "j"] \arrow[r, "k"']
\end{tikzcd}$$

then we can take the following for the back:

$$\begin{tikzcd}
a \arrow[r, "a"] & a & & \\
b \arrow[u, "p\ (\neg i)"] \arrow[r, "p\ (\neg k)"'] \arrow[ur, phantom, "p\ (\neg i \wedge \neg k)"] & a \arrow[u, "a"'] & & \bullet \arrow[u, "i"] \arrow[r, "k"']
\end{tikzcd}$$

This way we have constructed a cube with the top side missing where all faces match up and we can implement the above composition term. Note that we do not have to specify the ($k = 0$) face in the term, the reason is that when we substitute 0 for k we automatically get the term we want (remember that compPath p (sym p) is itself defined using comp).

A very important special case of the composition operation is the transport operation from Section 4. Given $\Gamma, i : \mathbb{I} \vdash A$ and $\Gamma \vdash a : A(0/i)$ we define

$$\mathsf{transport}^i\ A\ a := \mathsf{comp}^i\ A\ []\ a$$

The intuition behind this definition is exactly that transport is just composition with no fixed faces and it is in this sense that the composition operation generalizes transport. Furthermore, by defining comp by cases on all type formers in the theory we obtain the cases for $\mathsf{transport}$ from above as special cases. This means that we have to add computation rules for composition instead of transport to the theory and this is exactly what is done in [Coh+18, Section 4.5].

An important consequence of defining $\mathsf{transport}$ using composition is that we can now construct a path between a and $\mathsf{transport}^i\ A\ a$:

$$\mathsf{transFill} : (a : A(0/i)) \to \mathsf{Path}^i\ A\ a\ (\mathsf{transport}^i\ A\ a)$$
$$\mathsf{transFill}\ a = \lambda(j : \mathbb{I}).\ \mathsf{comp}^i\ A\ [(j = 0) \mapsto a]\ a$$

When j is 0 this computes to a and when j is 1 the face constraint can be deleted, giving the above definition of $\mathsf{transport}$ in terms of comp. Using connections we can also define a similar "filling" operation that lets us construct a path between an element and a composition with that element as base. Geometrically this corresponds to an operation that lets us fill the *interior* of an open cube, e.g. given an open square or cube as above the filling operation computes a filled square or cube with the original open square or cube as its boundary. This kind of operation is used to define the judgmental computation rules for composition in Σ- and Π-types, however we won't go into the details of this here and refer the interested reader to Cohen et al. [Coh+18].

In general, the composition operation looks as follows:

$$\mathsf{comp}^i\ A\ [(j_0 = \varepsilon_0) \mapsto a_0, \ldots, (j_n = \varepsilon_n) \mapsto a_n]\ b$$

Furthermore, in Cohen et al. [Coh+18] these satisfy the following properties:

(1) All faces are distinct (so duplicated faces can be removed from the list).
(2) Absurd faces (i.e. $(0 = 1)$ and $(1 = 0)$) can be removed from the list of faces.
(3) Permutations of faces in the list does not affect the resulting composition.
(4) If one of the faces is true (i.e. $(0 = 0)$ or $(1 = 1)$) then the whole composition reduces to the element at this face with 1 substituted for i.
(5) The faces in the list must match up pairwise.
(6) The base b must match up with each of the faces with 0 substituted for i.

These are a lot of properties to formulate and check. In order to make this more convenient Cohen et al. [Coh+18] introduced context restrictions:

$$\Gamma, (i = \varepsilon) \vdash \mathcal{J}$$

This is to be understood as $\mathcal{J}(\varepsilon/i)$. In other words, context restrictions lets us consider judgments on subfaces of cubes without having to apply substitutions. To check item (5) in the above list we check that

$$\Gamma, i : \mathbb{I}, (j_k = \varepsilon_k), (j_l = \varepsilon_l) \vdash a_k = a_l : A$$

for all pairs of k and l. This judgment hence ensures that the sides of the cube match up. Furthermore, to check that the sides match the base (property (6) above) we use the following judgment:

$$\Gamma, (j_k = \varepsilon_k) \vdash a_k(0/i) = b : A(0/i)$$

For more details how this works formally, both type theoretically and semantically, see Cohen et al. [Coh+18]. Note that other combinations of constraints are possible, for instance one does not have to delete absurd faces or one can consider composition problems where permutations matter. In fact, Angiuli et al. [AFH18] considers a variation of composition where none of properties (1)–(3) are satisfied which enables some optimizations in the way the composition operations compute in cartesian cubical type theory.

5.1. **Kan composition semantically.** Semantically these operations can be elegantly formulated using the internal language of the presheaf topos of cubical sets. Following Orton and Pitts [OP18] we can express this as additional *structure* on the types in the model. Proving that the type formers preserve this structure is the main part of the model construction and these proofs are very similar to the computation rules that we have formulated type theoretically in this paper and which are presented in detail for composition in [Coh+18, Section 4.3].

We can also understand these operations using categorical and homotopical language using lifting diagrams. In the categorical model we can view a type in context Γ as an element over Γ in the slice category $\widehat{\square}/\Gamma$ (or in some other cubical set category than $\widehat{\square}$). The semantic version of $\mathsf{transFill}\ A\ a$ corresponds to the diagonal in the diagram:

$$\begin{tikzcd}
1 \arrow[r] \arrow[d, "\delta_0"'] & A \arrow[d] \\
\mathbb{I} \arrow[r] \arrow[ur, dotted] & \Gamma
\end{tikzcd}$$

The top map corresponds to the element a as it's a point in A. The bottom map is not visible in the syntax, but it is implicit in the substitution principle for the judgment. The diagonal map then defines a path in A and the commutativity of the top triangle corresponds to the fact that the starting point of this path is a. This is exactly what the $\mathsf{transFill}$ operation gives us. The interested reader can consult [Ang+17, Section 1.2] for a more detailed discussion of this.

If we pretend that the above is instead in Top then we would say that the map on the right has the *right lifting property* with respect to the endpoint inclusion $\delta_0 : 1 \to [0,1]$, which in turn means that this map has the *path lifting property*. In other words, we can think of $\mathsf{transFill}$ as an operation that gives us a choice of solutions to path lifting problems.

We can also express the Kan filling operation diagrammatically. It corresponds to choice of a diagonal map as in the diagram below.

$$\begin{tikzcd}
\sqcup \arrow[r] \arrow[d, "\delta_0 \vee \varphi"'] & A \arrow[d] \\
\square \arrow[r] \arrow[ur, dotted] & \Gamma
\end{tikzcd}$$

In general the square and open square on the left can be n-dimensional, but in the diagram we have just drawn the special case used for path concatenation. The map on the left is an endpoint inclusion combined with a "formula" specifying the shape of the open box. This formula is essentially a large disjunction of the faces in the syntax for the filling problem (in the case of path concatenation it is $(j = 0) \vee (j = 1)$). The precise definition of how this map is then defined can be elegantly expressed using pushout-products as in [GS17, Section 2].

The composition operation can also be drawn using lifting diagrams. This is done by pulling back the diagram for the filling operation as in the diagram below. The composition operation then produces the longer dashed arrow from the "lid" of the square in:

$$\begin{tikzcd}
\cdot\ \cdot \arrow[r] \arrow[d] \arrow[dr, phantom, very near start, "\lrcorner"] & \sqcup \arrow[r, rightarrowtail] \arrow[d] & A \arrow[d] \\
- \arrow[r] \arrow[urr, dotted, rightarrowtail] & \square \arrow[r] \arrow[ur, dotted, rightarrowtail] & \Gamma
\end{tikzcd}$$

Readers familiar with Kan simplicial sets will see a close resemblance between the simplicial horn filling diagrams and these cubical open box filling diagrams. In fact, this analogy can be made precise by saying that a map is a *fibration* if it has the right lifting property with respect to maps of the form $\delta_0 \vee \varphi$ as above [GS17, Section 2]. Sattler [Sat17] has proved that these, combined with a class of cofibrations, forms a model structure. Interestingly this construction uses that the type formers preserve the composition structure, in particular that the universe can be equipped with a filling structure. This can be contrasted with the Kan simplicial set model where the existence of the classical Quillen model structure using Kan fibrations is assumed prior to constructing the model of the type theory. In the constructive cubical set models the order is the opposite: first the model of the type theory is constructed and using this Sattler [Sat17] constructs a model structure.

Another important difference between the cubical and simplicial models is that being "Kan" is a structure instead of a property. For this to work constructively additional "uniformity" conditions on the choice of fillers are assumed. These are expressed using suitable naturality conditions and essentially say that the chosen fillers commute with substitution. The fact that uniform fillers can be used to obtain a constructive model of HoTT/UF was one of the main new ideas in the original cubical set model of Bezem et al. [BCH14].

5.2. Variations on the Kan operations.

In this paper we have presented the composition operations of Cohen et al. [Coh+18], but there are many other possible variations. In order to be able to support higher inductive types Coquand et al. [CHM18] replaced the composition operations with a notion of *homogeneous* composition where the type is constant. However, this is not enough and the transport operations also had to be generalized to get a theory where the operations are interderivable with the heterogeneous composition operations of Cohen et al. [Coh+18].

A crucial property in Cohen et al. [Coh+18] is that Kan filling can be derived from composition and connections, however in the cartesian setting there are no connections and in order to overcome this problem the composition operations need to be generalized as in Angiuli et al. [AFH18; Ang+17]. There is also a decomposition of this generalized composition operation into generalized homogeneous composition and another generalized form of transport that is called "coercion". This makes it possible to also support higher inductive types in these models as proved by Cavallo and Harper [CH19].

There are hence a lot of parameters one can vary when constructing cubical models. This makes the literature on cubical models quite difficult to get into as different papers use different variations when talking about similar things. However, in a recent paper of Cavallo et al. [CMS20] a common generalization to all of the (structural) cubical models was presented, making it clearer how they are all related. By weakening the generalized composition operations of the cartesian model the authors construct a more general model that specializes to the specific models in presence of additional structure.

5.3. **Exercises.**

(1) Given $p : \mathsf{Path}\ A\ a\ b$, prove the following groupoid laws using compositions (hint: draw suitable open cubes)
   i. $\mathsf{compRefl} : \mathsf{Path}\ (\mathsf{Path}\ A\ a\ b)\ (\mathsf{compPath}\ p\ \mathsf{refl}_b)\ p$
   ii. $\mathsf{reflComp} : \mathsf{Path}\ (\mathsf{Path}\ A\ a\ b)\ (\mathsf{compPath}\ \mathsf{refl}_a\ p)\ p$
   iii. $\mathsf{symComp} : \mathsf{Path}\ (\mathsf{Path}\ A\ b\ b)\ (\mathsf{compPath}\ (\mathsf{sym}\ p)\ p)\ \mathsf{refl}_b$

(2) Prove associativity of $\mathsf{compPath}$ using only composition (harder).

(3) Using compositions we can construct various "constant" n-cubes.
   i. Construct a square that has "constantly" $p : \mathsf{Path}\ A\ a\ a$ as its boundary:

$$\begin{tikzcd} a \arrow[r, "p"] & a \\ a \arrow[u, "p"] \arrow[r, "p"'] & a \arrow[u, "p"'] \end{tikzcd}$$

   That is, construct a term of the following type given $p : \mathsf{Path}\ A\ a\ a$:

$$\mathsf{constSquare} : \mathsf{Path}^i\ (\mathsf{Path}\ A\ (p\ i)\ (p\ i))\ p\ p$$

   ii. Given $p : \mathsf{Path}\ A\ a\ b$ and $q : \mathsf{Path}\ A\ b\ c$ generalize the above square to a filler for

$$\begin{tikzcd} b \arrow[r, "q"] & c \\ a \arrow[u, "p"] \arrow[r, "p"'] & b \arrow[u, "q"'] \end{tikzcd}$$

   iii. Given $p, q : \mathsf{Path}\ A\ a\ a$, why is it impossible to find fillers for the following squares:

$$\begin{tikzcd} a \arrow[r, "p"] & a \\ a \arrow[u, "p"] \arrow[r, "q"'] & a \arrow[u, "q"'] \end{tikzcd} \qquad \begin{tikzcd} a \arrow[r, "p"] & a \\ a \arrow[u, "q"] \arrow[r, "p"'] & a \arrow[u, "q"'] \end{tikzcd}$$

   iv. Construct a cube that is constantly $p : \mathsf{Path}\ A\ a\ a$ on all of its edges. (hard)

(4) Given some type A, use a composition to prove:

$$\mathsf{isContrProp} : \mathsf{isContr}\ A \to \mathsf{isProp}\ A$$

For the definition of isContr see the beginning of Section 6.

(5) Use a composition to prove:

$$\mathsf{isPropIsContr} : \mathsf{isProp}\ (\mathsf{isContr}\ A)$$

(6) We can define $\mathsf{isSet} : \mathcal{U} \to \mathcal{U}$ that expresses that a type is an h-set as

$$\mathsf{isSet}\ A = (x\ y : A) \to (p\ q : \mathsf{Path}\ A\ x\ y) \to \mathsf{Path}\ (\mathsf{Path}\ A\ x\ y)\ p\ q$$

Use a composition to prove:

$$\mathsf{isPropSet} : \mathsf{isProp}\ A \to \mathsf{isSet}\ A$$

(7) Given $f : A \to B$ and $g : B \to A$ prove the following using a composition

$$\mathsf{isPropRetract} : ((x\ :\ A) \to \mathsf{Path}\ A\ (g\ (f\ x))\ x) \to \mathsf{isProp}\ B \to \mathsf{isProp}\ A$$

6. Glue types and univalence

We have now finally reached the main goal of this paper: the constructive proof of the univalence axiom. When expressed in cubical type theory the axiom says:

$$\mathsf{univalence} : (A\ B : \mathcal{U}) \to \mathsf{Equiv}\,(\mathsf{Path}\ \mathcal{U}\ A\ B)\,(\mathsf{Equiv}\,A\,B)$$

Where the type of equivalences is defined as:

$$\begin{aligned}
\mathsf{Equiv}\,A\,B &:= (e : A \to B) \times \mathsf{IsEquiv}\,e \\
\mathsf{IsEquiv}\,e &:= (x : B) \to \mathsf{IsContr}\,(\mathsf{Fiber}\,e\,x) \\
\mathsf{IsContr}\,C &:= (x : C) \times ((y : C) \to \mathsf{Path}\ C\ y\ x) \\
\mathsf{Fiber}\,e\,x &:= (y : A) \times \mathsf{Path}\ B\ (e\ y)\ x
\end{aligned}$$

It can proved the above formulation of univalence is equivalent to the following two terms:[9]

$$\mathsf{ua} : (A\ B : \mathcal{U}) \to \mathsf{Equiv}\,A\,B \to \mathsf{Path}\ \mathcal{U}\ A\ B$$

$$\mathsf{ua}_\beta : (A\ B : \mathcal{U})\ (e : \mathsf{Equiv}\,A\,B)\ (a : A) \to \mathsf{Path}\ B\ (\mathsf{transport}^i\ (\mathsf{ua}\ A\ B\ e\ i)\ a)\ (e.1\ a)$$

The naive way of trying to prove ua would be to just add it a constant with a suitable computation rule:

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash e : \mathsf{Equiv}\,A\,B}{\Gamma \vdash \mathsf{ua\ A\ B\ e} : Path\,\mathcal{U}\ A\ B}$$

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash e : \mathsf{Equiv}\,A\,B \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{transport}^i\ (\mathsf{ua}\ A\ B\ e\ i)\ a = e.1\ a : B}$$

This would make ua_β trivially provable using refl and we could prove univalence. But this does not completely solve the problem of giving univalence computational meaning as there is no rule for "$\mathsf{transport}^i\ (\mathsf{ua}\ A\ B\ e\ (\sim\ i))\ b$". We could of course add another computation rule having it compute to the inverse of e applied to b, but then what about "$\mathsf{transport}^i\ (\mathsf{compPath}\ (\mathsf{ua}\ A\ B\ e)\ (\mathsf{sym}(\mathsf{ua}\ A\ B\ e)))\ i)\ a$"?

[9]See https://groups.google.com/forum/#!msg/homotopytypetheory/j2KBIvDw53s/YTDK4D0NFQAJ for the original discussion of this.

The above argument shows that it is not enough to just naively add some computation rules to the type theory in order to give univalence computational meaning.[10] It also shows that the real difficulty with giving univalence computational meaning comes from explaining how $\mathsf{transport}$, or more generally comp, should compute for paths built up using ua. In order to solve this problem Cohen et al. [Coh+18] introduced a new type theoretic construct called Glue types. These resemble the compositions from above, but instead of replacing faces with n-dimensional cubes the Glue types lets us replace faces of a type with equivalent types.

Just as we used composition to change the sides of a line we can use Glue types to replace the sides of a line between types with equivalent types. Given $i : \mathbb{I} \vdash A$ and two types B_0 and B_1 with equivalences $e_0 : \mathsf{Equiv}\, B_0\, A(0/i)$ and $e_1 : \mathsf{Equiv}\, B_1\, A(1/i)$ we can illustrate the type

$$\mathsf{Glue}\, [(i = 0) \mapsto (B_0, e_0), (i = 1) \mapsto (B_1, e_1)]\, A$$

using the diagram:

$$\begin{tikzcd}
B_0 \arrow[r, dashed, "{\mathsf{Glue}\, [(i = 0) \mapsto (B_0, e_0), (i = 1) \mapsto (B_1, e_1)]\, A}"] \arrow[d, squiggly, no head, "e_0"'] & B_1 \arrow[d, squiggly, no head, "e_1"] \\
A(0/i) \arrow[r, "A"'] & A(1/i)
\end{tikzcd}$$

This explains why these types are called Glue types: they let us *glue* together the endpoints of the line A with B_0 and B_1 along equivalences e_0 and e_1.[11] Note that this diagram is fundamentally different from the diagrams we drew for compositions–the sides are not lines, they are equivalences. This is illustrated by the sides being squiggly arrows without any specified direction.

It is straightforward to prove that the identity function is an equivalence (see exercise (1) for a direct cubical proof of this) and we write $\mathsf{id}_A : \mathsf{Equiv}\, A\, A$ for this equivalence. Given types A and B in $\mathcal{U}$ with $e : \mathsf{Equiv}\, A\, B$ we can construct the ua term as follows:

$$\begin{aligned}
\mathsf{ua} &: (A\ B : \mathcal{U}) \to \mathsf{Equiv}\, A\, B \to \mathsf{Path}\ \mathcal{U}\ A\ B \\
\mathsf{ua}\ A\ B\ e\ &= \lambda(i : \mathbb{I}).\ \mathsf{Glue}\, [(i = 0) \mapsto (A, e), (i = 1) \mapsto (B, \mathsf{id}_B)]\ B
\end{aligned}$$

We can illustrate this diagrammatically as follows:

$$\begin{tikzcd}
A \arrow[r, dashed, "{\mathsf{Glue}\, [(i = 0) \mapsto (A, e), (i = 1) \mapsto (B, \mathsf{id}_B)]\, B}"] \arrow[d, squiggly, no head, "e"'] & B \arrow[d, squiggly, no head, "\mathsf{id}_B"] \\
B \arrow[r, "B"'] & B
\end{tikzcd}$$

[10]Note that this argument is not a proof that it is impossible to just add a ua constant with some computation rules, but rather that it is not sufficient to just naively add some rules which suggests that a more structured approach is necessary.

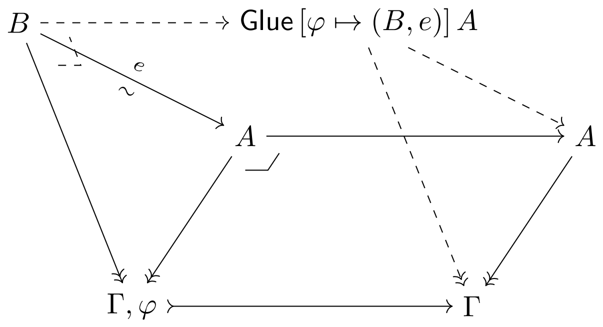
[11]There are a many different of notions of "*gluing*" in the literature, for instance the gluing axiom for sheaves or Artin gluing in topos theory. The notion of gluing considered here in the form of Glue types has no relationship with these other notions apart from the name.

We now have a definition of ua and the only missing part is to prove ua_β. For this we have to define transport, and more generally composition, for Glue types. Doing this is very complicated and the interested reader can look at [Coh+18, Section 6.2] for a complete definition. If one unfolds this definition in the special case of ua_β one almost get $e.1\ a$, except for some trivial transports in constant types that can be manually removed in order to construct the desired path. A formalization of this result and the standard formulation of the univalence axiom in Cubical Agda can be found at `https://github.com/agda/cubical/blob/master/Cubical/Foundations/Univalence.agda`.

As the above proof of univalence is a concrete definition in terms of the primitives of cubical type theory which all have computational content it as well has computational content. Furthermore, Huber [Hub16] has proved that this formulation of cubical type theory satisfies canonicity which means that any closed term of type $\mathbb{N}$ evaluates to a numeral. This hence provides constructive meaning to the univalence axiom. Furthermore, it is also possible to introduce identity types in cubical type theory and prove the univalence axiom expressed using them instead of paths (for a formal proof see `https://github.com/agda/cubical/blob/master/Cubical/Foundations/Id.agda`). This means that any term in HoTT/UF can be translated to cubical type theory and hence be given computational meaning.

6.1. **Glue types semantically.** Glue types can also be presented internally in the presheaf topos of cubical sets. This, together with a proof of a formulation of univalence without universes, was worked out by Orton and Pitts [OP18] and later extended to also incorporate universes by Licata et al. [Lic+18].

There is also a very similar construction to Glue types in the Kan simplicial set model [KL12, Theorem 3.4.1]. For a discussion of this see [Coh+18, Section 6.1] where the following diagram occurs:


$$\begin{tikzcd}
B \arrow[rr, dashed] \arrow[dr, "e", "\sim"'] \arrow[dd, two heads] \arrow[dr, phantom, "\lrcorner", very near start] & & \mathsf{Glue}\,[\varphi \mapsto (B,e)]\,A \arrow[dr, dashed] \arrow[dd, dashed, two heads] & \\
 & A \arrow[rr] \arrow[dl, two heads] \arrow[dr, phantom, "\lrcorner", very near start] & & A \arrow[dl, two heads] \\
\Gamma,\varphi \arrow[rr, tail] & & \Gamma &
\end{tikzcd}$$

The general form of this was coined the *Equivalence Extension Property* by S. Awodey and it plays an important role in the work of Sattler [Sat17] for constructing a model structure from a cubical model of HoTT/UF.

6.2. **Variations.** Just as there are differences in how the composition operations are formulated in different models, there are also variations in how the Glue types are formulated. It might seem strange to introduce the very general Glue types when we only want to be able to prove ua. There is indeed a simpler formulation in cartesian cubical sets which is tailored for proving ua and which Angiuli et al. [AFH18] call "V-types". These lets us omit the id_B side in the definition ua using Glue types and directly get the dashed line in:

$$\begin{tikzcd}
A \arrow[d, squiggly, dash, "e"'] \arrow[dr, dashed, "\mathsf{V}\ A\ B\ e"] & \\
B \arrow[r, "B"'] & B
\end{tikzcd}$$

Another, even simpler variation called G-types, can be found in Bezem et al. [BCH18]. However, this variation does not work in structural cubical models and type theories as explained by [Ang19, Page 65].

6.3. **Exercises.**

(1) Prove $\mathsf{id}_A : \mathsf{Equiv}\, A\, A$ using a connection.

(2) Given $f : A \to B$ and $g : B \to A$ prove the following using only compositions (hard):

$$\mathsf{isoToEquiv} : ((y : B) \to \mathsf{Path}\, B\, (f\ (g\ y))\ y) \to ((x : A) \to \mathsf{Path}\, A\, (g\ (f\ x))\ x) \to \mathsf{Equiv}\, A\, B$$

This lemma is very useful for constructing equivalences.

(3) Prove that $\mathsf{not} : \mathsf{bool} \to \mathsf{bool}$ from exercise (4) in Section 3 is an equivalence (hint: use $\mathsf{isoToEquiv}$). Combine this with ua to get a non-trivial path from bool to bool. What happens if we transport true along it?

(4) Use a 2-dimensional Glue type to prove

$$\mathsf{uaIdEquiv} : \mathsf{Path}\ (\mathsf{Path}\ \mathcal{U}\ A\ A)\ (\mathsf{ua}\ \mathsf{id}_A)\ \mathsf{refl}_A$$

7. Conclusions and further reading

We end these lecture notes with some pointers to further reading about cubical methods in HoTT/UF that hasn't already been discussed in the paper. These pointers are in no way meant to be an exhaustive list of the interesting literature on cubical methods in HoTT/UF, but rather just some pointers to papers that an interested reader can look into next.

For the reader who wants more hands-on experience with the ideas presented in these notes we recommend trying out Cubical Agda [VMA19]. There are also a few papers reporting on formalization projects performed using this system, including synthetic homotopy theory [MP20], proof theory and ordinal notations [FXG20], and a formalization of π-calculus [VV20].

There are also some interesting papers describing various independence results that have been proved using cubical methods. For instance, Uemura [Uem18] used a cubical variation of assemblies to construct an impredicative universe that does not satisfy a form of propositional resizing, Coquand et al. [CMR17] showed that countable choice cannot be proved in univalent type theory with propositional truncation, and the independence of Church's thesis in univalent type theory was shown by Swan and Uemura [SU19].

As explained in these notes cubical type theory is constructive and hence satisfies the existence property. This means that we can write down existence statements using Σ-types and extract witnesses automatically. A famous example of this is the so called *"Brunerie number"*: a concrete synthetic definition of $n \in \mathbb{Z}$ such that $\pi_4(\mathbb{S}^3) = \mathbb{Z}/n\mathbb{Z}$ [Bru16]. This construction has been formalized Cubical Agda, but it has so far not been possible to compute this numeral due to the computational complexity of the involved constructions. This is hence a very important open problem and solving it would lead to the possibility of constructively computing many other topological invariants using cubical type theory.

Finally, an important problem with cubical type theory is the question whether we can interpret all of the results that we prove in topological spaces or even any (Grothendieck) ∞-topos. Currently these questions have not been fully resolved for the various cubical type theories and models that we have discussed in this paper. However, there has been some recent progress on an "equivariant" cubical set model that is equivalent to spaces. We are hence very optimistic that these issues will be resolved in the near future, leading to even more exciting developments in HoTT/UF using cubical methods.

References

[AFH18] Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. "Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities". In: *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*. 2018, 6:1–6:17.

[Ang+17] Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. "Syntax and Models of Cartesian Cubical Type Theory". Draft available at `https://github.com/dlicata335/cart-cube/blob/master/cart-cube.pdf`. 2017.

[Ang+18] Carlo Angiuli, Evan Cavallo, Kuen-Bang Hou (Favonia), Robert Harper, and Jonathan Sterling. "The RedPRL Proof Assistant (Invited Paper)". In: *13th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2018)*. Ed. by Frédéric Blanqui and Giselle Reis. Vol. 274. Electronic Proceedings in Theoretical Computer Science. Oxford, UK, 2018.

[Ang19] Carlo Angiuli. "Computational Semantics of Cartesian Cubical Type Theory". PhD thesis. Carnegie Mellon University, 2019.

[Awo18] Steve Awodey. "A cubical model of homotopy type theory". In: *Annals of Pure and Applied Logic* 169.12 (2018). Logic Colloquium 2015, pp. 1270–1294.

[BCH14] Marc Bezem, Thierry Coquand, and Simon Huber. "A Model of Type Theory in Cubical Sets". In: *19th International Conference on Types for Proofs and Programs (TYPES 2013)*. Vol. 26. Leibniz International Proceedings in Informatics (LIPIcs). 2014, pp. 107–128.

[BCH18] Marc Bezem, Thierry Coquand, and Simon Huber. "The Univalence Axiom in Cubical Sets". In: *Journal of Automated Reasoning* (June 2018).

[BCP15] Marc Bezem, Thierry Coquand, and Erik Parmann. "Non-Constructivity in Kan Simplicial Sets". In: *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*. Ed. by Thorsten Altenkirch. Vol. 38. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015, pp. 92–106.

[BM17] Ulrik Buchholtz and Edward Morehouse. "Varieties of Cubical Sets". In: *Relational and Algebraic Methods in Computer Science*. Ed. by Peter Höfner, Damien Pous, and Georg Struth. Springer International Publishing, 2017, pp. 77–92.

[Bru16] Guillaume Brunerie. "On the homotopy groups of spheres in homotopy type theory". PhD thesis. Université de Nice, 2016.

[CH19] Evan Cavallo and Robert Harper. “Higher Inductive Types in Cubical Computational Type Theory”. In: *Proc. ACM Program. Lang.* 3.POPL (Jan. 2019), 1:1–1:27.

[CHM18] Thierry Coquand, Simon Huber, and Anders Mörtberg. “On Higher Inductive Types in Cubical Type Theory”. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS ’18. ACM, 2018, pp. 255–264.

[CMR17] Thierry Coquand, Bassel Mannaa, and Fabien Ruch. “Stack semantics of type theory”. In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. June 2017, pp. 1–11.

[CMS20] Evan Cavallo, Anders Mörtberg, and Andrew W Swan. “Unifying Cubical Models of Univalent Type Theory”. In: *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*. Ed. by Maribel Fernández and Anca Muscholl. Vol. 152. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, 14:1–14:17.

[Coh+18] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. “Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom”. In: *Types for Proofs and Programs (TYPES 2015)*. Vol. 69. LIPIcs. 2018, 5:1–5:34.

[Con+85] R. L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development Environment*. Prentice-Hall, 1985.

[cubical] The cubical Development Team. *cubical*. Available at `https://github.com/simhu/cubical/`. 2013–.

[cubicaltt] The cubicaltt Development Team. *cubicaltt*. Available at `https://github.com/mortberg/cubicaltt/`. 2015–.

[Dyb96] Peter Dybjer. “Internal Type Theory”. In: *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Heidelberg, New York, 1996, pp. 120–134.

[FXG20] Fredrik Nordvall Forsberg, Chuangjie Xu, and Neil Ghani. “Three Equivalent Ordinal Notation Systems in Cubical Agda”. In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2020. New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 172–185.

[GS17] Nicola Gambino and Christian Sattler. “The Frobenius condition, right properness, and uniform fibrations”. In: *Journal of Pure and Applied Algebra* 221.12 (2017), pp. 3027–3068.

[Hof97] Martin Hofmann. “Syntax and semantics of dependent types”. In: Publications of the Newton Institute 14 (1997). Ed. by Andrew M. Pitts and Peter Dybjer, pp. 79–130.

[HS97] Martin Hofmann and Thomas Streicher. “Lifting Grothendieck Universes”. Unpublished Note. 1997.

[Hub16] Simon Huber. “Canonicity for Cubical Type Theory”. Preprint arXiv:1607.04156 [cs.LO]. July 2016.

[KL12] Chris Kapulkin and Peter LeFanu Lumsdaine. “The Simplicial Model of Univalent Foundations (after Voevodsky)”. Preprint arXiv:1211.2851v4 [math.LO]. Nov. 2012.

[Lic+18] Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. “Internal Universes in Models of Homotopy Type Theory”. In: *FSCD*. Vol. 108.

LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 22:1–22:17.

[Mar75] Per Martin-Löf. “An Intiutionistic Theory of Types: Predicative Part”. In: *Logic Colloquium ’73*. Ed. by H. E. Rose and J. Shepherdson. North–Holland, Amsterdam, 1975, pp. 73–118.

[Mar82] Per Martin-Löf. “Constructive Mathematics and Computer Programming”. In: *Logic, Methodology and Philosophy of Science, VI*. 1982, pp. 153–175.

[Mar84] Per Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.

[Mar98] Per Martin-Löf. “An intuitionistic theory of types”. In: *Twenty-five years of constructive type theory (Venice, 1995)*. Vol. 36. Oxford Logic Guides. New York: Oxford Univ. Press, 1998, pp. 127–172.

[MP20] Anders Mörtberg and Loïc Pujet. “Cubical Synthetic Homotopy Theory”. In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2020. New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 158–171.

[OP18] Ian Orton and Andrew M. Pitts. “Axioms for Modelling Cubical Type Theory in a Topos”. In: *Logical Methods in Computer Science* Volume 14, Issue 4 (Dec. 2018).

[Par14] Jason Parker. “Duality between Cubes and Bipointed Sets”. MA thesis. Carnegie Mellon University, 2014.

[Red16] The RedPRL Development Team. *The RedPRL Proof Assistant*. Available at http://www.redprl.org/. 2016–2018.

[Red18] The RedPRL Development Team. *redtt*. Available at https://github.com/RedPRL/redtt. 2018.

[Sat17] Christian Sattler. “The Equivalence Extension Property and Model Structures”. Preprint arXiv:1704.06911v1 [math.CT]. 2017.

[SU19] Andrew Swan and Taichi Uemura. *On Church’s Thesis in Cubical Assemblies*. 2019.

[Uem18] Taichi Uemura. “Cubical Assemblies and Independence of the Propositional Resizing Axiom”. Preprint arXiv:1803.06649 [cs.LO]. 2018.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: http://homotopytypetheory.org/book, 2013.

[VMA19] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. “Cubical Agda: A Dependently Typed Programming Language with Univalence and Higher Inductive Types”. Preprint available at http://www.cs.cmu.edu/ amoertbe/papers/cubicalagda.pdf. 2019.

[Voe10] Vladimir Voevodsky. “Univalent Foundations Project”. A modified version of an NSF grant application. Oct. 2010.

[Voe11] Vladimir Voevodsky. “Univalent Foundations”. Plenary lecture at WoLLIC, May 18. 2011.

[Voe14] Vladimir Voevodsky. “The equivalence axiom and univalent models of type theory. (Talk at CMU on February 4, 2010)”. Preprint arXiv:1402.5556 [math.LO]. 2014.

[Voe15] Vladimir Voevodsky. “An experimental library of formalized Mathematics based on the univalent foundations”. In: *Mathematical Structures in Computer Science* 25 (2015), pp. 1278–1294.

[VV20] Niccolò Veltri and Andrea Vezzosi. "Formalizing Pi-Calculus in Guarded Cubical Agda". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2020. New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 270–283.

[yacctt] The yacctt Development Team. *yacctt*. Available at `https://github.com/mortberg/yacctt/`. 2018.