Mathematical Structures in Computer Science (2021), **31**, pp. 1147–1184
doi:10.1017/S0960129521000311

CAMBRIDGE UNIVERSITY PRESS

PAPER

Cubical methods in homotopy type theory and univalent foundations

Anders Mörtberg*

Department of Mathematics, Stockholm University, Stockholm, Sweden
*Corresponding author. Email: anders.mortberg@math.su.se

(Received 8 March 2020; revised 10 September 2021; accepted 14 September 2021; first published online 10 December 2021)

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 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.

Keywords: Homotopy type theory; univalent foundations; cubical type theory; cubical set models

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 paper is hence 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 (2014) is provable and has computational content. This can therefore be seen as a fully constructive foundation for HoTT/UF, well-suited for computer implementation.

The univalence axiom of Voevodsky (2014) is at the center for HoTT/UF and provides a type theoretic rendering of the idea that all constructions should be invariant up to some form of equivalence. This is a 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. It is also common in computer science to think of two equivalent implementations of an abstract interface as the same. 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 (Univalent Foundations Program, 2013; Voevodsky, 2015). 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 loses 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 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 (Kapulkin and Lumsdaine, 2012). However, this model used classical logic in crucial ways (Bezem et al., 2015) which led experts to look in different directions for a constructive model. The first major breakthrough was when Bezem et al. (2014) formulated the first *cubical* model of HoTT/UF which constituted the starting point of the use of cubical methods in HoTT/UF.

This paper introduces cubical methods in HoTT/UF by focusing on the cubical type theory developed by Cohen et al. (2018). 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. (2014). 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 and type theories based on cartesian cubical sets (Angiuli et al., 2021*a*, 2018b; Awodey, 2018). For a more comprehensive historical account of the various cubical set models and type theories, we refer the interested reader to Angiuli et al. (2021*a*, Section 1).

Outline. The paper 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), 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), and finally, in order to be able to prove the univalence axiom and give it computational content Glue types are introduced (Section 6). The paper then concludes with some suggestions for further reading about cubical methods in HoTT/UF (Section 7). Furthermore, most sections end with exercises that complement the material and which were given to the students during the summer school. Suggested solutions to most exercises can be found in Appendix A.

Assumed background knowledge. These lecture notes are aimed at students and researchers with some familiarity with dependent type theory and category theory. There are not many textbooks on type theory out there; however, a classic is the book by Nordström et al. (1990). Another good introduction which goes into the categorical semantics of type theory is the notes by Hofmann (1997). The situation for category theory is much better as it is an older field. These notes do not require that much categorical background apart from familiarity with presheaves and the Yoneda lemma. An excellent introduction to these concepts with lots of examples can be found in the book

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

by Riehl (2017, Chapter 1 and 2.1–2.2). In order to get a deeper understanding of lifting problems and the content of Section 5.4, see Riehl (2014, Chapter 11), but beware that this requires much more categorical background than the rest of the material and are not necessary unless one wants to study the homotopy theoretical aspects of the field.

2. Background

Homotopy Type Theory and Univalent Foundations, as formulated by Voevodsky (2010, 2011, 2014, 2015) and in the HoTT book (Univalent Foundations Program, 2013), are axiomatic extensions of type theory initially developed by the Swedish logician Per Martin-Löf. There are multiple different type theories of this kind, see, for example, Martin-Löf (1975, 1982, 1984, 1998), 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 $\Gamma \vdash \mathscr{J}$ 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. We follow the `Agda` convention (introduced by Nuprl Constable et al. 1985) 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 $p : (x : A) \times B$. We also assume Π- and Σ-types satisfy judgmental η rules:

$$\frac{\Gamma \vdash f : (x : A) \to B}{\Gamma \vdash f = \lambda(x : A).fx : (x : A) \to B} \qquad \frac{\Gamma \vdash p : (x : A) \times B}{\Gamma \vdash p = (p.1, p.2) : (x : A) \times B}$$

We say that an equality holds "judgmentally" if it is a judgmental equality specified by the type theory, that is, if the judgment $\Gamma \vdash a = b : A$ is derivable in the theory. Note that many authors refer to this as "definitional equality"; however, we prefer the term "judgmental equality" as it is more specific to the MLTT formalism. We will refer to the semantic version of this judgment as a "strict equality," that is, an equality that holds strictly in some model of the theory.

We also assume an infinite hierarchy of universes $\mathscr{U}_n$; 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 data types like empty, unit, Boolean, and natural number types are also assumed and inhabit the lowest universe $\mathscr{U}_0$. We will write functions on these using pattern-matching equations; however, all examples can easily be translated to recursors and eliminators. One can also imagine extending with general inductive types or W-types, but we will not focus on these here as the cubical machinery extends very straightforwardly from the basic inductive types and it is more illustrative to look at concrete examples instead of the general case. However, the situation for inductive families (a.k.a. indexed inductive types) is more subtle and we refer the interested reader to Cavallo and Harper (2019) and Vezzosi et al. (2021, Section 4) for details.

However, we do not include Martin-Löf (1975) 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 functions $\mathbb{I} \to A$ with fixed endpoints. 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 cubical 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.

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

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` (2013), `cubicaltt` (2015), `yacctt` (2018), `RedPRL` (2016) (Angiuli et al., 2018*a*), `redtt` (2018), `cooltt` (2020), `mlang` (2019), and `Cubical Agda` (Vezzosi et al., 2019). All of these systems build on different cubical type theories and have different standard cubical models (Angiuli et al. 2021*a*, 2018b; Bezem et al. 2014, 2019; Cavallo and Harper 2019; Cohen et al. 2018; Coquand et al. 2018); 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. 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 (1996). 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 data types like natural numbers, together with universes closed under these type formers. These results can be found in the lecture notes of Hofmann (1997) and the unpublished note about universes by Hofmann and Streicher (1997).

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 (2018). This approach builds on an idea of Coquand (2015), and another early use of the technique can be found in the work of Birkedal et al. (2019). As these categories are locally cartesian closed, this internal language is extensional type theory (Lambek and Scott, 1986), 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. For the strict extensional equality in the internal language, we write $x \equiv y$ given $x, y : A$. 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*, that is, present the semantics using standard categorical language. We will in the paper make it clear when the semantics is described internally or externally.

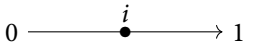
The main aim of these notes is to present cubical type theories and their models up to the computational realization of univalence. The cubical version of this principle informally states that the type of equivalences of types A and B is equivalent to the type of paths between A and B. This in particular means that we have a function

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

Combined with cubical transport, this gives a way to transport structures between equivalent types. For example, if we have a monoid structure on unary numbers we can transport it to get a monoid structure on binary numbers. The fact that univalence has computational meaning means that this transport will actually compute as expected.

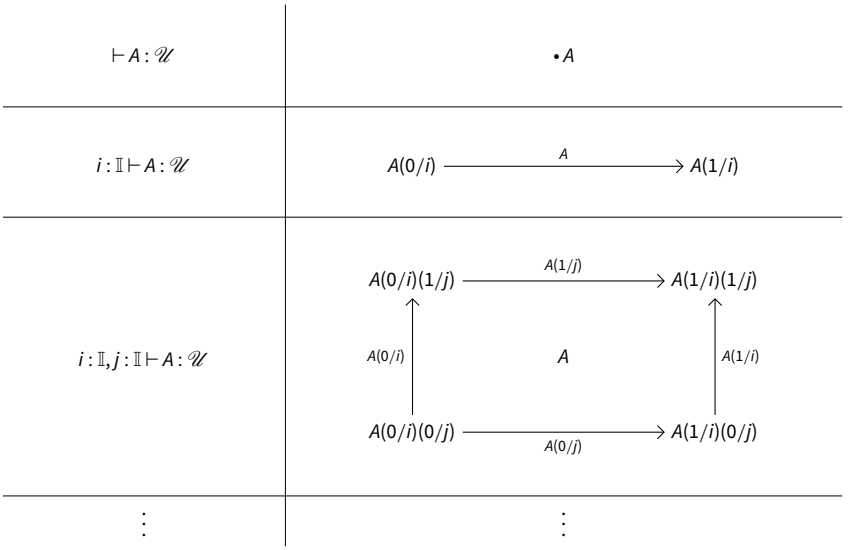
3. Cubical Type Theories and Their Models

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 $1 : \mathbb{I}$:



https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

Table 1. Relationship between contexts with interval variables and cubes



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

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

Given a judgment $\mathscr{J}$ depending on an interval variable $i : \mathbb{I}$, we write $\mathscr{J}(r/i)$ for $\mathscr{J}$ 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 in Table 1.

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, $A(0/j)$, in the third row of the Table 1.

Remark 1. Note that any judgment in cubical type theory can be put in the above form. Indeed, given a judgment $\Gamma \vdash \mathscr{J}$ where Γ contains both interval variables $i : \mathbb{I}$ and regular variables $x : A$ in any order, we can always first reorganize it so that the interval variables occur first and the regular variables occur last (weakening appropriately with respect to the interval variables). The regular variables can then be λ-abstracted so that we only have interval variables left in the context. Focusing on the cubical judgments of the above form is hence not a restriction, and as it simplifies the explanations, we will focus on them in this section; however, in later sections Γ denotes an arbitrary context with both interval and regular variables unless explicitly specified.

3.1 Cartesian cubical sets

Semantically, all cubical set models are based on presheaves on some cube category $\mathscr{C}$, that is, functor categories $\widehat{\mathscr{C}} := [\mathscr{C}^{\mathrm{op}}, \mathbf{Set}]$. This means that a cubical set $G : \widehat{\mathscr{C}}$ is simply a functor $\mathscr{C}^{\mathrm{op}} \to \mathbf{Set}$ and hence comes with both an action on the objects of $\mathscr{C}^{\mathrm{op}}$ (which are the same as those of $\mathscr{C}$) and the morphisms of $\mathscr{C}^{\mathrm{op}}$ (which are the same as those in $\mathscr{C}$, but reversed). So given an element of $X : \mathscr{C}$ we get a set $G(X)$ and given a morphism $f : \mathrm{Hom}_{\mathscr{C}}(X, Y)$, we get a function

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

of sets $G(f) : G(Y) \to G(X)$. As G is a functor, it furthermore has to send the identity morphism $1_X : \mathrm{Hom}_{\mathscr{C}}(X, X)$ to the identity function $1_{G(X)} : G(X) \to G(X)$ and composition of morphisms $g \circ f$ to composition of functions $G(f) \circ G(g)$.

One way to think about presheaves is as indexed sets with extra structure. Indeed, a special case is when $\mathscr{C}$ is some small discrete category I, that is, a category where the collection of objects is a set and the only morphisms are the identity morphisms. A presheaf $G : \widehat{I}$ is then nothing but an indexed family of sets (indexed by the objects of I). Now, if $\mathscr{C}$ happens to not be discrete and have more morphisms than the identity morphisms then we get induced maps between the indexed sets. Presheaves are hence a compact way of describing a collection sets indexed by the objects of a category, together with functions between them respecting composition. So in order to specify a cubical set model, we first have to give a suitable cube category specifying what to index our cubical sets by and which functions we should be able to manipulate these with.

A very important cube category is the *cartesian* one.

Definition 2. *The* cartesian *cube category* $\square$ *has as objects finite sets, and as morphisms* $\mathrm{Hom}_{\square}(I, J)$ *functions* $J \to I + \{0, 1\}$ *with identity and composition given by*

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

The reason I and J are flipped in the functions in the above definition is that we will take the opposite of this category when defining cartesian cubical sets. In the rest of the notes, we will omit the $\mathsf{inl}/\mathsf{inr}$ maps for readability as they can easily be inferred. Furthermore, note that the composition operation is the Kleisli composition for the monad $\_ + \{0, 1\} : \square \to \square$. 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 above.

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

- Given $\varepsilon \in \{0, 1\}$, a dimension i and a finite set I there are *face maps*

$$d^i_\varepsilon \in \mathrm{Hom}_{\square}(I, I + \{i\})$$
$$d^i_\varepsilon(j) = \begin{cases} \varepsilon & \text{if } i = j \\ j & \text{otherwise} \end{cases}$$

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

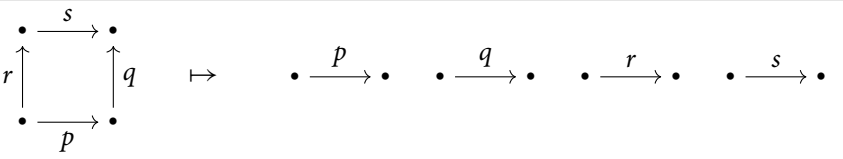
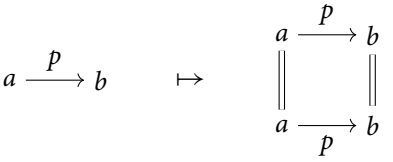
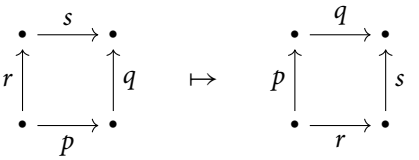
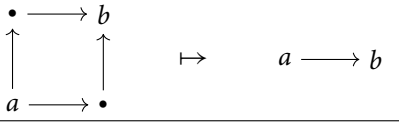
$$s^i \in \mathrm{Hom}_{\square}(I + \{i\}, I)$$
$$s^i(j) = j$$

- Given dimensions i, j, and a finite set I, there are *symmetry maps*

$$t^{i,j} \in \mathrm{Hom}_{\square}(I + \{i, j\}, I + \{i, j\})$$
$$t^{i,j}(k) = \begin{cases} j & \text{if } k = i \\ i & \text{if } k = j \\ k & \text{otherwise} \end{cases}$$

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

Table 2. Faces, degeneracies, symmetries and diagonals

Faces	
Degeneracies	
Symmetries	
Diagonals	

- Given dimensions i, j, and a finite set I, there are *diagonal maps*

$$c^{i,j} \in \mathrm{Hom}_{\square}\,(I + \{i\}, I + \{i, j\})$$

$$c^{i,j}(k) = \begin{cases} i & \text{if } k = i \text{ or } k = j \\ k & \text{otherwise} \end{cases}$$

These morphisms satisfy various evident cubical identities, for example, degenerating and taking a face does nothing. We illustrate the actions of these maps informally for concrete n-cubes in Table 2.

A cartesian cubical set is a functor $G : \widehat{\square}$. Geometrically, we may think of such a G as a space and $G(\{i_1, \ldots, i_n\})$ as the set of continuous functions $[0, 1]^n \to G$. With this in mind, we can give some geometric meaning to the action of G on the morphisms of $\square$. The face maps $G(d^i_\varepsilon) : G(I + \{i\}) \to G(I)$ restrict $(n+1)$-cubes to n-cubes by setting the i coordinate to ε. The degeneracy maps $G(s^i) : G(I) \to G(I + \{i\})$ 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.

Let $\text{ⱶ} : \square \to \widehat{\square}$ be the Yoneda[3] embedding defined at an object $I \in \square$ as $\text{ⱶ}(I) = \mathrm{Hom}_{\square}\,(\_, I)$. At a morphism $f : \mathrm{Hom}_{\square}\,(I, J)$, it is defined as precomposition of f with $g : \mathrm{Hom}_{\square}\,(\_, I)$, so $\text{ⱶ}(f)(g) = f \circ g$. The interval of cubical type theory is modeled by $\text{ⱶ}(\{i\})$ where i is an arbitrary dimension. We write $\mathbb{I}$ for this representable 1-cube (a representable cubical set is one which is in the image of ⱶ). An important property of cartesian cubical sets is that the product of representables is again representable.[4] Combined with the fact that ⱶ preserves products, we get that the representable n-cube is an n-fold product of $\mathbb{I}$, that is, $\text{ⱶ}(\{i_1, \ldots, i_n\}) \cong \mathbb{I}^n$. The category $\widehat{\square}$ is in fact the universal category with this property:

Remark 3. The category $\widehat{\square}$ is equivalent to the free finite product category on an interval object (Awodey, 2018; Parker, 2014).

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

The Yoneda lemma is probably the most used result in category theory. In the case of a cartesian cubical set G, it states that we have a bijection $G(\{i_1, \ldots, i_n\}) \cong \mathbb{I}^n \to G$. In other words, the n-cubes of G are in bijective correspondence with the set of natural transformations $\mathbb{I}^n \to G$. This means that the structure of a cartesian cubical set is determined by maps out of products of $\mathbb{I}$, justifying the geometric intuition from above where we thought of G as a space and $G(\{i_1, \ldots, i_n\})$ as the set of continuous functions $[0,1]^n \to G$. Furthermore, a type $i_1 : \mathbb{I}, \ldots, i_n : \mathbb{I} \vdash A : \mathscr{U}$ corresponds to a morphism $A : \mathbb{I}^n \to \mathscr{U}$, justifying the cubical judgments. This is one reason why cartesian cubical sets are so well-suited as a basis for higher dimensional type theory.

3.2 Cubical set models

It is a well-known fact that any presheaf category forms a model of type theory (Hofmann, 1997, Section 4). This hence directly applies to the cubical set category $\widehat{\square}$ discussed above. In this model, a context Γ is modeled by a cartesian cubical set G and a substitution $\sigma : \Delta \to \Gamma$ is modeled by a natural transformation of cubical sets $s : D \to G$. A type in context Γ is also modeled by a presheaf, but on *the category of elements* of G. The category of elements, written $\int G$, is a general construction on a presheaf category which gives a new category where the objects are of the form (I, ρ) with $I : \square$ and $\rho : G(I)$. A type is hence modeled by a presheaf on $\int G$. With these definitions, one can prove that $\widehat{\square}$ can be organized into a CwF with structure corresponding to Π-, Σ-types, basic data types and universes closed under these operations. This means in particular that we have a context extension operation which given presheaves $G : \widehat{\square}$ and $A : \widehat{\int G}$ gives us an extended semantic context $G.A : \widehat{\square}$ as well as a weakening map $p : G.A \to G$. Terms of the type theory are then modeled as sections of p.

Remark 4. An important, but subtle, fact is that by working with types represented as objects of $\widehat{\int G}$, we avoid coherence problems related to equations between substitutions on types and terms only holding up to isomorphism. A classical result in category theory is that $\widehat{\int G} \simeq \widehat{\square}/G$, so presheaves on the category of elements of G is equivalent to the slice category over G. Types could hence just as well be modeled by objects of $\widehat{\square}/G$, but this has the drawback that substitution would be defined as a pullback which is only unique up to isomorphism. This means that if we would substitute with $\sigma : \Delta \to \Gamma$ and then $\delta : \Theta \to \Delta$ in A, we would not get *strictly* the same thing as if we would substitute with $\delta \circ \sigma : \Theta \to \Gamma$ directly. This is a well-known problem which arises in models of type theory in locally cartesian closed categories (Seely, 1984) and various solutions exist to solve it (Clairambault and Dybjer, 2011; Curien, 1993; Curien et al., 2014; Hofmann, 1994; Lumsdaine and Warren, 2015; Voevodsky, 2009). However, in the case of presheaf categories like $\widehat{\square}$ this problem can completely be avoided by instead representing types as objects in $\widehat{\int G}$ where substitution can simply be defined as precomposition and hence satisfy the required equations strictly by definition.

After this short introduction to presheaf models of type theory, we can now clarify the relationship between the proof theory of cubical type theory and its semantics. Given an n-cube $I = \{i_1, \ldots, i_n\}$ and context $\Gamma = i_1 : \mathbb{I}, \ldots, i_n : \mathbb{I}$, there is a close correspondence between the structural rules of cubical type theory and the semantic maps of cubical sets as illustrated in Table 3.

Recall that the action of $\curlywedge$ on morphisms is precomposition, so the judgments on the left are given by precomposing with the maps on the right. The table shows that there is a close correspondence between the maps in the cartesian cubical set categories and the structure of the cubical judgments in cubical type theory. This correspondence was analyzed by Buchholtz and Morehouse (2017) to define and relate a great variety of cubical set categories.

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

Table 3. Relationship between structural rules and semantic maps

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

Remark 5. If we omit contraction/diagonals, we obtain the *substructural* cubical sets that underlie the original cubical set model of Bezem et al. (2014). 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). It also not clear how to interpret eliminators for HITs in this cubical set model. In the rest of the paper, we hence focus on structural cubical sets with contraction/diagonals.

3.3 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.

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

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

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

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. So just like how Π-types "internalize" the usual variables of type theory, the path types let us internalize the interval variables of cubical type theory.

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, that is, paths living over a line of types. These path-over types are descendants of the work of McBride (2002) on heterogeneous equality in type theory (i.e., equalities between terms of different types). Having built-in path-over types is very useful,

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

especially when working with higher inductive types, as we can directly represent heterogeneous equalities. Furthermore, representing equalities using path types allows direct definitions of many standard operations on identity types that are usually proved by path induction.

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

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

We will in the rest of paper sometimes write just refl when the a can be easily inferred.

Remark 6. 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 $A, B : \mathscr{U}$ 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 (omitting the a and b arguments):

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

The fact that we get these equations judgmentally is very useful when formalizing mathematics. Note that the standard equation $\mathsf{ap}\ f\ (\mathsf{refl}\ a) = \mathsf{refl}\ (f\ a)$ also holds judgmentally as $\mathsf{refl}\ a$ is just a constant function.

We can also define new operations that does not always hold for identity types, for instance, function extensionality for path types can be proved as:

$$\mathsf{funExt} : (f\ g : A \to B)\ (p : (x : A) \to \mathsf{Path}\ B\ (f\ x)\ (g\ x)) \to \mathsf{Path}\ (A \to B)\ f\ g$$
$$\mathsf{funExt}\ f\ g\ p = \lambda(i : \mathbb{I}).\ \lambda(x : A).\ p\ x\ i$$

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 judgmental 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.

Even though we can define a funExt constant, it is often more convenient to inline it. In HoTT/UF, propositions are defined as types satisfying a predicate that says that all elements are path-equal:

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

A basic result of HoTT/UF, which uses function extensionality, is that propositions are closed under Π-types.

$$\mathsf{isPropPi} : (A : \mathscr{U})\ (B : A \to \mathscr{U})\ (h : (x : A) \to \mathsf{isProp}\ (B\ x)) \to \mathsf{isProp}\ ((x : A) \to B)$$
$$\mathsf{isPropPi}\ A\ B\ h = \lambda\ (f\ g : (x : A) \to B)\ (i : \mathbb{I})\ (x : A).\ h\ x\ (f\ x)\ (g\ x)\ i$$

Note that we have inlined the use of funExt here. In fact, we are implicitly using funExt for dependent functions in order to prove this, but with path types there is no difference in the proofs of the dependent and non-dependent version. For details see exercise (5).

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

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

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

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. In particular, the inverse of Σ_{eq} can be defined as

$$\Sigma^{-}_{\mathsf{eq}} : (s\ t : (x : A) \times B)\ (q : \mathsf{Path}\ ((x : A) \times B)\ s\ t) \to (p : \mathsf{Path}\ A\ s.1\ t.1) \times (\mathsf{Path}^i\ (B\ (p\ i))\ s.2\ t.2)$$
$$\Sigma^{-}_{\mathsf{eq}}\ s\ t\ q = (\lambda(i : \mathbb{I}).\ (q\ i).1, \lambda(i : \mathbb{I}).\ (q\ i).2)$$

The fact that Σ_{eq} and Σ^{-}_{eq} are mutually inverse is proved by refl as we have judgmental β and η rules for the involved types. Being able to pass seamlessly between paths in Σ-types and Σ-types of paths is an easily overlooked benefit of having path types.

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

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

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.

3.4 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 done in both `cubicaltt` and `Cubical Agda` which are based on the cube category of the CCHM cubical set model of Cohen et al. (2018). Before defining this category, we have to introduce De Morgan algebras:

Definition 7. *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 8. *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 are inherited from the Kleisli category of* DM. *A* CCHM cubical set *is a functor $G : \widehat{\square}_{\mathsf{DM}}$.*

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

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

Table 4. Connections and reversals

Connections	$a \xrightarrow{p} b$	$\mapsto$	$\begin{array}{ccc} a & \xrightarrow{p} & b \\ \| & \wedge & \uparrow p \\ a & = & a \end{array}$ $\quad$ $\begin{array}{ccc} b & = & b \\ p \uparrow & \vee & \| \\ a & \xrightarrow[p]{} & b \end{array}$
Reversals	$a \longrightarrow b$	$\mapsto$	$b \longrightarrow a$

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

$$c^k_{i\wedge j} \in \mathrm{Hom}_{\square_{\mathsf{DM}}}\ (I + \{i,j\}, I + \{k\}) \qquad c^k_{i\vee j} : \mathrm{Hom}_{\square_{\mathsf{DM}}}\ (I + \{i,j\}, I + \{k\})$$

$$c^k_{i\wedge j}(l) = \begin{cases} i \wedge j & \text{if } k = l \\ l & \text{otherwise} \end{cases} \qquad c^k_{i\vee j}(l) = \begin{cases} i \vee j & \text{if } k = l \\ l & \text{otherwise} \end{cases}$$

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

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

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

The connections can be thought of as new kinds of degeneracies while the reversals let us invert lines as illustrated in Table 4.

The action of the reversal map is, as the name suggests, to reverse a path. The connections, on the other hand, might look a bit funny at first. However, if one thinks of them as new types of degeneracies they are less strange. As discussed above, the usual degeneracy map s^i can be thought of as an operation which turns a path p from a to b into a square:

$$\begin{array}{ccc} a & \xrightarrow{p} & b \\ \| & & \| \\ a & \xrightarrow[p]{} & b \end{array}$$

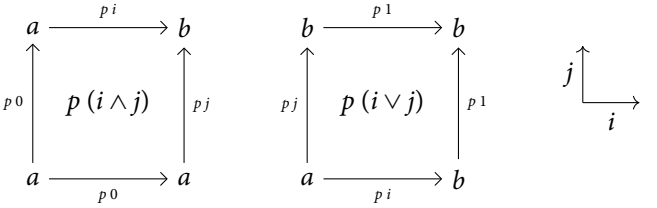
Reading this as a commutative diagram of paths, this represents a proof that $\mathsf{refl} \cdot p = p \cdot \mathsf{refl}$ (where $\cdot$ is composition in diagrammatic order). The connections analogously represent proofs that $\mathsf{refl} \cdot p = \mathsf{refl} \cdot p$ and $p \cdot \mathsf{refl} = p \cdot \mathsf{refl}$. This might not seem very useful, but note that we have not yet defined general composition of paths. So by adding operations to the cube category, we get the possibility of constructing fillers to certain squares representing compositions directly. We will soon see that this lets us prove some results about path types which would normally require path induction. Furthermore, combining connections with reversals allows us to fill many squares, for example, the one corresponding to $p \cdot p^{-1} = \mathsf{refl} \cdot \mathsf{refl}$ (see exercise (6)).

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 $r, s \in [0, 1]$.

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

Remark 9. One might wonder why De Morgan algebras and not 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 $r \in [0, 1]$ with the above interpretation 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



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).

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

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 $\lambda(j : \mathbb{I}).\ p\ j$, that is, from refl 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:

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

This satisfies $\mathsf{sym}\ (\mathsf{sym}\ p) = p$ judgmentally. This is another example of an equation that does not 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 $\mathscr{C}^{\mathrm{op}^{\mathrm{op}}} = \mathscr{C}$ 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 (1975).

Exercises

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

(2) Given $a, a' : A$ and $b, b' : B$, prove the binary version of ap:

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

(3) Given $f, g : A \to B \to C$, prove the binary (non-dependent) version of funExt:

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

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

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

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

(5) Given dependent functions $f, g : (x : A) \to B$, prove funExt for dependent functions:

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

Use this to give an alternative proof of isPropPi where funExtDep is used explicitly.

(6) Given $p : \mathsf{Path}\ A\ a\ b$ and $i, j : \mathbb{I}$, draw the squares corresponding to
   i. $p\ (\neg i \wedge j)$
   ii. $p\ (i \wedge \neg j)$
   iii. $p\ (\neg i \wedge \neg j)$
   iv. $p\ (\neg i \vee j)$
   v. $p\ (i \vee \neg j)$
   vi. $p\ (\neg i \vee \neg j)$

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

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

(9) Given $A : \mathscr{U}$, prove the following variation of contrSingl:

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

(10) 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 $\lambda(i\,j : \mathbb{I}).\ \mathsf{loop}\ (i \wedge j)$ constructs a square whose $(i = 0)$ and $(i = 1)$ sides are refl base and loop? (**Hint:** what are the other sides of the square?)

(11) As $i \wedge \neg i$ is not necessarily 0 in a De Morgan algebra, we can construct a path that goes halfway around the circle and back by writing $\lambda(i : \mathbb{I}).\ \mathsf{loop}\ (i \wedge \neg i)$ (recall that $i \wedge \neg i$ corresponds to $\min(i, 1 - i)$). Construct a homotopy on the circle that shows that this path is contractible:

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

(12) 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. Cubical Transport

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

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

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:

$$a \bullet \qquad\qquad \bullet\ \mathsf{transport}^i\ A\ a$$

$$A(0/i) \xrightarrow{\quad A \quad} A(1/i)$$

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

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 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.

4.1 Recovering HoTT/UF transport and path induction

Readers familiar with HoTT/UF will probably be surprised by the type signature of cubical transport. It is indeed different from what is called "transport" in HoTT/UF. The HoTT/UF transport operation takes a type family $P : A \to \mathscr{U}$, a path $a = b$ in A, and gives a function $P\ a \to P\ b$. The cubical transport operation on the other hand takes a line $A : \mathbb{I} \to \mathscr{U}$ and gives a function $A(0/i) \to A(1/i)$. This more primitive transport operation does however imply the HoTT/UF transport operation, which we refer to as "subst" in order to avoid confusion:

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

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

With this, we can now prove the path induction principle J. Indeed, it may in fact be decomposed as contractibility of singletons (which we have proved in the previous section) and subst:

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

$$\mathsf{J}\ A\ a\ C\ d\ x\ p =$$

$$\mathsf{subst}\ ((x : A) \times \mathsf{Path}\ A\ a\ x)\ (\lambda\ (y : (x : A) \times \mathsf{Path}\ A\ a\ x).\ C\ y.1\ y.2)$$

$$(a, \mathsf{refl})\ (x, p)\ (\mathsf{contrSingl}\ A\ a\ x\ p)\ d$$

With this, we can now do similar reasoning as in HoTT/UF. For example, path composition and its properties can be derived from J. Furthermore, we can represent structures on types using Σ-types (or records if we would have them) and transport properties between structured types.

Example 4.1. We can define what it means for a type to be a monoid as a nested Σ-type as follows:

$$\mathsf{isMonoid} : \mathscr{U} \to \mathscr{U}$$

$$\begin{aligned}\mathsf{isMonoid}\ A = {} & (e : A) \\ & \times (op : A \to A \to A) \\ & \times (id : (a : A) \to \mathsf{Path}\ A\ (op\ e\ a)\ a \times \mathsf{Path}\ A\ (op\ a\ e)\ a) \\ & \times (assoc : (a\ b\ c : A) \to \mathsf{Path}\ A\ (op\ a\ (op\ b\ c))\ (op\ (op\ a\ b)\ c)) \\ & \times (\mathsf{isSet}\ A)\end{aligned}$$

The first four fields are the identity constant and binary operation of the monoid together with their standard laws. The final field makes sure that the type is a set in the HoTT/UF sense, that is, its path type is a proposition (see exercise (6) for the formal definition). Using subst, we can now get a function for transporting monoid structure between path-equal types:

$$\mathsf{subst}\ \mathscr{U}\ \mathsf{isMonoid} : (A\ B : \mathscr{U}) \to \mathsf{Path}\ \mathscr{U}\ A\ B \to \mathsf{isMonoid}\ A \to \mathsf{isMonoid}\ B$$

Concretely, A could be unary natural numbers $\mathbb{N}$ and B binary numbers $\mathbb{B}$. This means that we can transport the monoid structure on $\mathbb{N}$ to $\mathbb{B}$ given a path between these two types. We will see that such a path can be constructed using univalence in Example 6.1. This hence means that we will get an induced associative and unital binary operation on binary numbers from the one on unary numbers.

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

With what we have seen so far we can hence formalize many results of HoTT/UF; however, we still have not given these notions computational meaning. For example, how should the transported monoid operation on binary numbers compute? It is easy to imagine how it *could* compute, but as the underlying cubical transport operation has no judgmental computation rules yet this function would be stuck when given concrete binary numbers. Furthermore, as P in subst can be any predicate valued in $\mathscr{U}$ and not just isMonoid we need to give judgmental computation rules to cubical transport which handle all of the type formers of the theory.

Another problem is that it is not yet possible to prove the standard computation rule for path induction which says that we have a path:

$$\mathsf{Path}\ (C\ a\ \mathsf{refl})\ (\mathsf{J}\ A\ a\ C\ d\ a\ \mathsf{refl})\ d$$

Indeed, if we unfold the left-hand side we see that we have to construct a path:

$$\mathsf{Path}\ (C\ a\ \mathsf{refl})\ (\mathsf{transport}^{i}\ (C\ a\ \mathsf{refl})\ d)\ d$$

However, there is currently no reason why this would be the same as d for all type families C. One could try to add a computation rule which says that $\mathsf{transport}^{i}\ A\ a$ is judgmentally a if i does not occur A, but this does not help with making the monoid example above compute properly. It is in fact currently an open problem whether the above path can be proved by $\mathsf{refl}\ d$ while also having well-behaved computation rules for the rest of the theory. This is commonly referred to as the "regularity problem" in cubical type theory and constructive cubical models. A solution to this problem would be a major breakthrough, but there are some negative counterexamples by Swan (2018) which suggest that it might in fact be impossible to achieve this.

All currently existing cubical type theories and constructive cubical models circumvent this problem by not requiring the above path to be provable by $\mathsf{refl}\ d$, but rather carefully setting things up so that it can be proved by a slightly more involved path. Achieving this while still giving well-behaved computation rules to the theory is really the main technical part of constructing a cubical type theory and constructive cubical models. We will now look at how this can be achieved for cubical transport for some basic type formers.

4.2 Judgmental computation rules for cubical transport

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

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

It is easy to see that this is a well-defined judgmental 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 $\mathsf{transport}^{\neg i}\ A\ a : A(0/i)$:

$$\mathsf{transport}^{\neg i}\ A\ a\ \bullet \qquad\qquad \bullet\ a$$

$$A(0/i) \xrightarrow{\quad A \quad} A(1/i)$$

Using a reversal we can define this as:

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

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

We can now give the definition of cubical transport for $A \to B$. Given $\Gamma, i : \mathbb{I} \vdash A \to B : \mathscr{U}$ 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:

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

For basic data types A without parameters, we can let $\mathsf{transport}^i\ A$ be the identity function, for example, 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 data types (like $A + B$ or $\mathsf{List}\ A$), we can define the cubical transport operation for each constructor (see exercise (2)). This generalizes directly to W-types and schemas for inductive types. However, as discussed in Section 2, the situation for inductive families is more subtle and has recently been worked out for a type theory similar to the one in these notes by Vezzosi et al. (2021, Section 4) by building on work by Cavallo and Harper (2019).

Remark 10. Note that there is a choice in how cubical transport behaves for pair and function types. We could instead have treated these transports as neutral values that do not 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 cubical 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 cubical transport for Π- and Σ-types (see exercise (3)). The way we solve this, following Cohen et al. (2018), is to generalize cubical transport and instead consider more general *composition* operations.

Exercises

(1) Give the *negative* definition of cubical transport for $A \times B$.
(2) Define cubical transport in sum types $A + B$. (**Hint:** pattern-match on the constructors)
(3) Try to define cubical transport for Σ- and Π-types and see what problems you run into.

5. Kan Composition Operations

In order to solve the problem with cubical transport for path types, we introduce a generalized cubical transport operation that also lets us fix the boundary of the transported element. Given

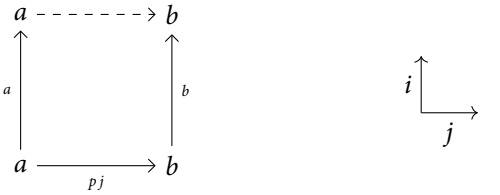
https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

$p : (\mathsf{Path}\ A\ a\ b)(0/i)$, we write this as:

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

This is a path in $A(1/i)$ where the boundary has been fixed to be a and b. The syntax of the comp operation is the same as the one for cubical 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 cubical transport where we can fix the sides of the transported element. We can illustrate the above example diagrammatically as follows:



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") have 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 "valid" (i.e., turn the equation before "$\mapsto$" into $(0 = 0)$ or $(1 = 1)$) the operation reduces to the corresponding element at this face with 1 substituted for i. So if we substitute 0 for j in

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

we will get

$$\mathsf{comp}^i\ A\ [(0 = 0) \mapsto a, (0 = 1) \mapsto b]\ (p\ 0)$$

This a composition with a valid face which means that it will reduce to the element at this face with 1 substituted for i. In this case, the element at the face is a and we get

$$\mathsf{comp}^i\ A\ [(0 = 0) \mapsto a, (0 = 1) \mapsto b]\ (p\ 0) = a(1/i) = a$$

Furthermore, "absurd" faces with $(0 = 1)$ and $(1 = 0)$ before the "$\mapsto$" can be disregarded/deleted. So if there would not have been a valid face in the above composition, we could have just deleted the $(0 = 1) \mapsto b$ face. Finally, note that this operations binds i in all of A, a and b, but not in $p\ j$. This means that the reduction when a face is valid could trigger further computations as we will reduce the element at the valid face with 1 for i.

5.1 Examples of compositions

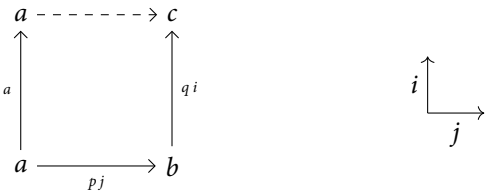
Let us now look at some examples of compositions. Given $a, b, c : A$, we can define the concatenation of two paths as:

$$\mathsf{compPath} : (p : \mathsf{Path}\ A\ a\ b)\ (q : \mathsf{Path}\ A\ b\ c) \to \mathsf{Path}\ A\ a\ c$$

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

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

This can be illustrated as the dashed line in:



Note that the right-hand side depends on i in this example, so if we apply compPath p q to 1 we will substitute 1 for j in the composition and get:

$$\mathsf{comp}^i\ A\ [(1=0)\mapsto a, (1=1)\mapsto q\ i]\ (p\ 1) = (q\ i)(1/i) = q\ 1 = c$$

By the same discussion as the example above, we get a when applying compPath p q to 0. This hence means that compPath p q is indeed a path from a to c as specified in the type.

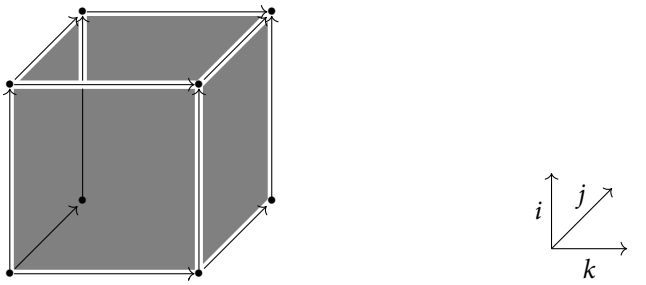
Let us now prove one of the groupoid laws: the concatenation of a path with its inverse is path-equal to refl. Given $a, b : A$ we give the following complicated definition:

$$\mathsf{compSym} : (p : \mathsf{Path}\ A\ a\ b) \to \mathsf{Path}\ (\mathsf{Path}\ A\ a\ a)\ (\mathsf{compPath}\ p\ (\mathsf{sym}\ p))\ \mathsf{refl}$$

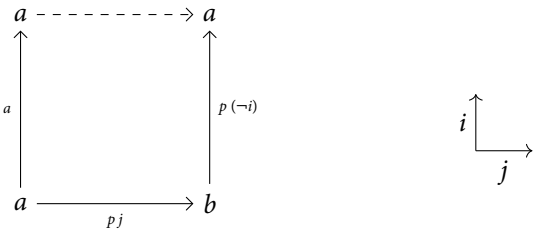
$$\mathsf{compSym}\ p = \lambda(k\ j : \mathbb{I}).\ \mathsf{comp}^i\ A\ [(j=0)\mapsto a, (j=1)\mapsto p\ (\neg i \wedge \neg k), (k=1)\mapsto a]\ (p\ (j \wedge \neg k))$$

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 absurd faces. Up to renaming of bound variables this is the same as compPath p (sym p) as desired. In the second case, the third face becomes valid (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 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, that is, a square, we will have to form a 3-dimensional composition drawing, that is, a cube. We draw this as follows:

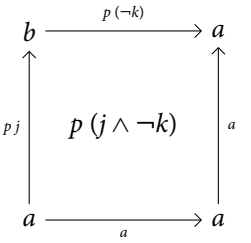


The top face is the desired result, that is, a square in direction k with left-hand side being compPath p (sym p) and all three other sides refl. This fixes the left $(k=0)$ side to:

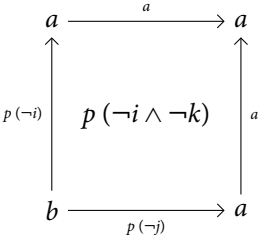


https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

For the front ($j = 0$) and right ($k = 1$) 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



then we can take the following for the back:



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).

5.2 Cubical transport as special case of composition

A very important special case of the composition operation is the cubical 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 that cubical transport is just composition with no faces fixed, and it is in this sense that the composition operation generalizes cubical transport. Furthermore, by defining comp by cases on all type formers in the theory we obtain the cases for cubical transport from above as special cases. This means that we have to add computation rules for composition instead of cubical transport to the theory and this is exactly what is done in Cohen et al. (2018, Section 4.5).

An important consequence of defining cubical transport using composition is that given type $A, B : \mathscr{U}$ we can now construct a path between $a : A$ and $\mathsf{transport}^i\ (P\ i)\ a$ over $P : \mathsf{Path}\ \mathscr{U}\ A\ B$:

$$\mathsf{transFill} : (P : Path\ \mathscr{U}\ A\ B)\ (a : A) \to \mathsf{Path}^i\ (P\ i)\ a\ (\mathsf{transport}^i\ (P\ i)\ a)$$

$$\mathsf{transFill}\ P\ a = \lambda(j : \mathbb{I}).\ \mathsf{comp}^i\ (P\ (i \wedge j))\ [(j = 0) \mapsto a]\ a$$

When j is 0, this computes to a and when j is 1 the face can be deleted, giving the above definition of cubical transport in terms of comp. With this we can now justify that the computation rule for J on refl holds up to a path. Recall that J $A\ a\ C\ d\ a$ refl reduces to $\mathsf{transport}^i\ (C\ a\ \mathsf{refl})\ d$,

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

so transFill does indeed give us the result we want (up to symmetry):

$$\begin{aligned}&\mathsf{JPath} : (A : \mathscr{U})\ (a : A)\ (C : (x : A) \to \mathsf{Path}\ A\ a\ x \to \mathscr{U})\ (d : C\ a\ \mathsf{refl}) \to \\ &\qquad \mathsf{Path}\ (C\ a\ \mathsf{refl})\ (\mathsf{J}\ A\ a\ C\ d\ a\ \mathsf{refl})\ d \\ &\mathsf{JPath}\ A\ a\ C\ d = \mathsf{sym}\ (\mathsf{transFill}\ (\mathsf{refl}\ (C\ a\ \mathsf{refl}))\ d)\end{aligned}$$

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, for example, 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 will not go into the details of this here and refer the interested reader to Cohen et al. (2018, Section 4.4 and 4.5).

5.3 Composition in general

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. (2018) 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 valid (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. (2018) introduced context restrictions:

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

This is to be understood as $\mathscr{J}\,(\varepsilon/i)$. In other words, context restrictions lets us consider judgments on faces 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:

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

With the notations of Cohen et al. (2018), the above discussion is summarized by the following concise typing rule:

$$\frac{\Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I} \vdash A \qquad \Gamma, \varphi, i : \mathbb{I} \vdash u : A \qquad \Gamma \vdash a_0 : A(0/i)[\varphi \mapsto u(0/i)]}{\Gamma \vdash \mathsf{comp}^i\ A\ [\varphi \mapsto u]\ a_0 : A(1/i)[\varphi \mapsto u(1/i)]}$$

The notation $\Gamma \vdash a : A[\varphi \mapsto b]$ is short for $\Gamma \vdash a : A$ and $\Gamma, \varphi \vdash a = b : A$, so a has type A and on φ it is judgmentally equal to b. This hence lets us very concisely formulate properties (5) and (6) in the typing rule. For more details how this works formally, both type theoretically and semantically,

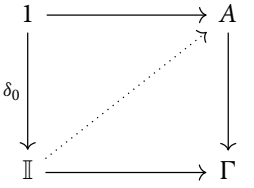
https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

see Cohen et al. (2018). 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. (2018*b*) consider 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.

5.4 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 (2018), 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 Cohen et al. (2018).

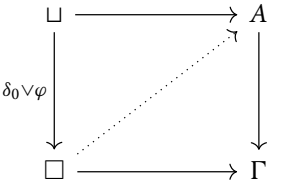
We can also understand these operations using categorical and homotopical language through lifting diagrams. In the categorical model, we can view a type in context Γ as an element over Γ in the slice category $\widehat{\square}_{\mathsf{DM}} / \Gamma$ (or in some other cubical set category than $\widehat{\square}_{\mathsf{DM}}$).[5] The semantic version of transFill A a corresponds to the diagonal in the diagram:



The top map corresponds to the element a as it is 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 dashed 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 transFill operation gives us. The interested reader can consult (Angiuli et al., 2021*a*, Section 1.2) for a more detailed discussion of this.

If we pretend that the above is instead in the category of topological spaces, 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 transFill as an operation that gives us a choice of solutions to path lifting problems.

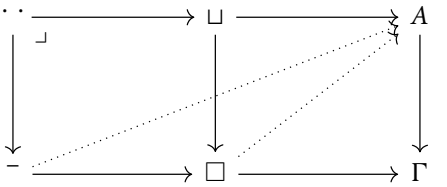
We can also express the Kan filling operation diagrammatically. It corresponds to choice of a diagonal map as in the diagram below.



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 Gambino and Sattler (2017, Section 2).

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

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:



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 (Gambino and Sattler, 2017, Section 2). Sattler (2017) has proved that these, combined with a class of cofibrations, forms a Quillen model structure. That is, a category suitable for developing homotopy theory in. 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 (2017) 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. (2014).

5.5 Variations on the Kan operations

In these notes, we have presented the composition operations of Cohen et al. (2018), but there are many other possible variations. In order to be able to support higher inductive types, Coquand et al. (2018) replaced the composition operations with a notion of *homogeneous* composition where the type is constant. However, this is not enough and the cubical 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. (2018).

A crucial property in Cohen et al. (2018) 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. (2021*a*, 2018b). There is also a decomposition of this generalized composition operation into generalized homogeneous composition and another generalized form of cubical transport called "coercion". This makes it possible to also support higher inductive types in these models as proved by Cavallo and Harper (2019).

There are hence a lot of parameters one can vary when constructing cubical models. This makes the literature on these models quite difficult to get into as different papers use different variations when talking about similar things. However, in a recent paper Cavallo et al. (2020) develop a common generalization to all of the (structural) cubical models, 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.

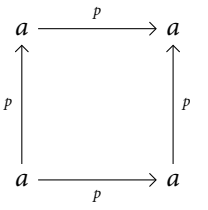
https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

Exercises

(1) Given p : Path A a b, prove the following groupoid laws using compositions (**Hint:** draw suitable open cubes)
   i. compRefl : Path (Path A a b) (compPath p refl) p
   ii. reflComp : Path (Path A a b) (compPath refl p) p
   iii. symComp : Path (Path A b b) (compPath (sym p) p) refl

(2) Prove associativity of compPath using only composition (harder).

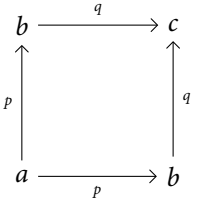
(3) Using compositions we can construct various "constant" n-cubes.
   i. Construct a square that has "constantly" p : Path A a a as its boundary:



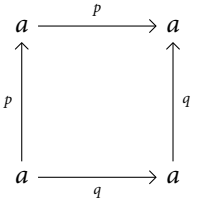
   That is, construct a term of the following type given p : Path A a a:

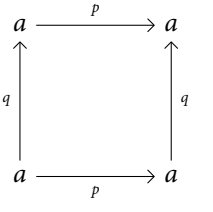
   constSquare : Path^i (Path A (p i) (p i)) p p

   ii. Given p : Path A a b and q : Path A b c generalize the above square to a filler for



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





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

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

   isContrProp : isContr A → isProp A

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

(5) Use a composition to prove:

   isPropIsContr : isProp (isContr A)

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

   isSet A = (x y : A) → (p q : Path A x y) → Path (Path A x y) p q

   Use a composition to prove:

   isPropIsSet : isProp A → isSet A

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

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

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

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:

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

where the type of equivalences is defined as:

$\mathsf{Equiv}\ A\ B = (e : A \to B) \times \mathsf{IsEquiv}\ e$

$\mathsf{IsEquiv}\ e = (x : B) \to \mathsf{IsContr}\ (\mathsf{Fiber}\ e\ x)$

$\mathsf{IsContr}\ C = (x : C) \times ((y : C) \to \mathsf{Path}\ C\ y\ x)$

$\mathsf{Fiber}\ e\ x = (y : A) \times \mathsf{Path}\ B\ (e\ y)\ x$

It can be proved that the above formulation of univalence is equivalent to the following two terms:[6]

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

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

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

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

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

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 "$\mathsf{transport}^i\ (\mathsf{compPath}\ (\mathsf{ua}\ A\ B\ e)\ (\mathsf{sym}(\mathsf{ua}\ A\ B\ e)))\ i)\ a$"?

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. It also shows that the real difficulty with giving univalence computational meaning comes from explaining how cubical transport, or more generally comp, should compute for paths built up using ua. 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.

6.1 Glue types

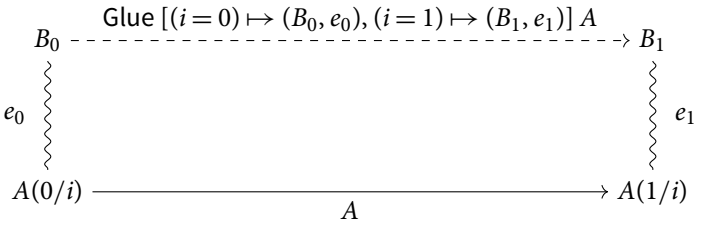
A more structured approach was introduced by Cohen et al. (2018) who invented a new type theoretic construct called "Glue types." These resemble the compositions from above, but instead of replacing faces with n-dimensional cubes the Glue types allows faces of a type to be replaced by equivalent types. 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

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

and B_1 with equivalences $e_0 : \mathsf{Equiv}\ B_0\ A(0/i)$ and $e_1 : \mathsf{Equiv}\ B_1\ A(1/i)$ not relying on $i : \mathbb{I}$, we can illustrate the type (in context $i : \mathbb{I}$)

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

using the diagram:

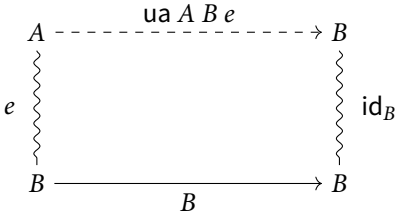


This explains why these types are called Glue types: they let us *glue* together the endpoints of the line A with B_0 and B_1 along equivalences e_0 and e_1.[7] Note that this diagram is fundamentally different from the diagrams we drew for compositions – the sides are not lines, they are equivalences. 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 $\mathscr{U}$ with $e : \mathsf{Equiv}\ A\ B$ we can construct the ua term as follows:

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

We can illustrate this diagrammatically as follows:



We now have a definition of ua and the only missing part is to prove ua_β. For this, we have to define cubical transport, and more generally composition, for Glue types. Doing this is very complicated, and the interested reader can look at Cohen et al. (2018, Section 6.2) for a complete definition. If one unfolds this definition in the special case of ua_β one almost gets $e.1\ a$, except for some trivial transports in constant types. This is hence exactly like what happened when we proved the computation rule for J at refl, and we can again use transFill to prove ua_β.

With ua and ua_β, we can now prove the standard formulation of univalence from above. To do this note that ua_β states that $\mathsf{Equiv}\ A\ B$ is a *retract* of $\mathsf{Path}\ \mathscr{U}\ A\ B$. It is easy to see that this implies that $(X : \mathscr{U}) \times \mathsf{Equiv}\ A\ X$ is a retract of $(X : \mathscr{U}) \times \mathsf{Path}\ \mathscr{U}\ A\ X$. But $(X : \mathscr{U}) \times \mathsf{Path}\ \mathscr{U}\ A\ X$ is contractible by contrSingl and any retract of a contractible type is itself contractible, so $(X : \mathscr{U}) \times \mathsf{Equiv}\ A\ X$ is in fact contractible. This is another way to state the univalence axiom and by Univalent Foundations Program (2013, Theorem 5.8.4) we get the desired proof of univalence. A formalization of these results in `Cubical Agda` can be found at https://github.com/agda/cubical/blob/ef62b84397396d48135d73ba7400b71c721ddc94/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.

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

Furthermore, Huber (2019) 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/ef62b84397396d48135d73ba7400b71c721ddc94/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.

With all of this, we can now finally transport arbitrary properties and structures between equivalent types. Let us spell this out for monoids.

Example 6.1. Combining ua and subst, we get a convenient function for transporting results between equivalent types:

$$\mathsf{substEquiv} : (P : \mathscr{U} \to \mathscr{U})\ (A\ B : \mathscr{U})\ (e : \mathsf{Equiv}\ A\ B)\ \to P\ A \to P\ B$$
$$\mathsf{substEquiv}\ P\ A\ B\ e\ x = \mathsf{subst}\ \mathscr{U}\ P\ A\ B\ (\mathsf{ua}\ e)\ x$$

In the case when P is isMonoid, this gives us a way of transporting monoid structures between related types as discussed in Example 4.1. A classic result in HoTT/UF is that isomorphic types are equivalent (see exercise (2)). Unary and binary numbers are clearly isomorphic, and we can hence use substEquiv to transport the additive monoid structure from unary to binary numbers. This means that we will get an addition operation on binary numbers induced by the one on unary numbers. As we have given computational meaning to cubical transport at all type formers involved, we can see that this operation will add two binary numbers by translating them to unary, adding them using unary addition, and then translating back to binary. This back and forth actually follows from the way cubical transport is defined for function types (remember that we had to transport "backwards" in the definition).

Such an addition operation on binary numbers might seem quite naive, but it turns out that we can actually do much more using univalence. The key realization is that univalence extends to a general principle called the *structure identity principle* (SIP) (Univalent Foundations Program, 2013, Section 9.8) which says that any property can be transported between equivalent *structured* types. We can define the type of monoids as

$$\mathsf{Monoid} = (A : \mathscr{U}) \times \mathsf{isMonoid}\ A$$

The SIP then lets us lift a structured equivalence on the underlying types of $M, N : \mathsf{Monoid}$ to a path between M and N. Combining this with subst we can now transport any property proved for M to N. We can hence prove that binary numbers with efficient binary addition is a monoid and relate this to the additive monoid on unary numbers. This way we achieve a clear *separation of concerns* (Dijkstra, 1974) where proofs can be done using unary numbers and then transported over to binary numbers which are better suited for doing computations than proving properties. A cubical version of the SIP and many of its consequences for formalizing mathematics and computer science can be found in the paper of Angiuli et al. (2021*b*).

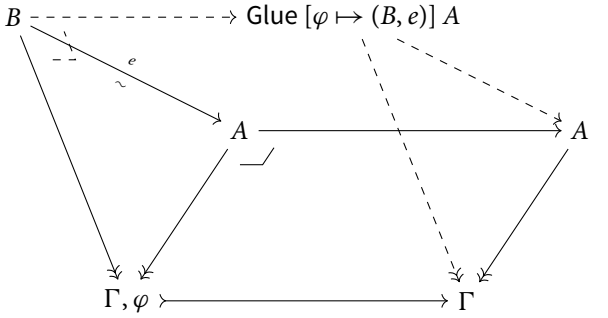
6.2 Glue types semantically

Glue types can also be presented internally in the presheaf topos of cubical sets we've seen in these notes. This, together with a proof of a formulation of univalence without universes, was worked out by Orton and Pitts (2018) and later extended to also incorporate universes by Licata et al. (2018). The typing rule for Glue types can be expressed as:

$$\frac{\Gamma \vdash A \qquad \Gamma, \varphi \vdash B \qquad \Gamma, \varphi \vdash e : \mathsf{Equiv}\ A\ B}{\Gamma \vdash \mathsf{Glue}\ [\varphi \mapsto (B, e)]\ A}$$

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

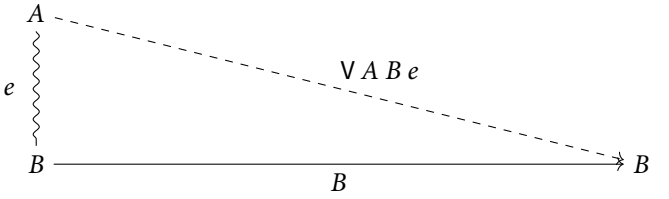
This can be expressed diagrammatically as follows:



The diagram states the same thing as the typing rule for Glue types: given a type A defined on all of Γ and a type B which is equivalent to the restriction of A on Γ, φ the Glue type gives us a total type defined on all of Γ. Interestingly, there is a very similar construction in the proof of univalence in the Kan simplicial set model (Kapulkin and Lumsdaine, 2012, Theorem 3.4.1) where a more or less identical diagram occurs. For a discussion of the relationship between these, see Cohen et al. (2018, Section 6.1). The general form of this was coined as the *Equivalence Extension Property* by Awodey and plays an important role in the work of Sattler (2017) for constructing a Quillen model structure from a cubical model of HoTT/UF.

6.3 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. (2018*b*) call "V-types." These lets us omit the id_B side in the definition of ua using Glue types and directly get the dashed line in:



However, this variation is weaker than Glue types in that they don't let us directly equip the universe with a composition structure. To remedy this, Angiuli et al. (2018*b*) introduce another quite complicated type of compositions in the universe which essentially correspond to a special case of Glue types.

Another, even simpler variation called G-types, can be found in Bezem et al. (2019). However, this variation does not work in structural cubical models and type theories as explained by Angiuli (2019, p. 65).

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.

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

(3) Prove that $\mathsf{not} : \mathsf{bool} \to \mathsf{bool}$ from exercise (4) in Section 3 is an equivalence (**Hint:** use 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

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

7. Conclusions and Further Reading

We end these lecture notes with some pointers to further reading about cubical methods in HoTT/UF that has not 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` (Vezzosi et al., 2019, 2021). A tutorial to `Cubical Agda` with many exercises can be found at https://github.com/HoTT/EPIT-2020/tree/main/04-cubical-type-theory. There are also quite a few papers reporting on formalization projects using `Cubical Agda`, including a cubical version of the SIP (Angiuli et al., 2021*b*), synthetic homotopy theory (Mörtberg and Pujet, 2020), proof theory and ordinal notations (Forsberg et al., 2020), and a formalization of π-calculus (Veltri and Vezzosi, 2020).

There are also some interesting papers describing various independence results that have been proved using cubical methods. For instance, Uemura (2019) used a cubical variation of assemblies to construct an impredicative universe that does not satisfy a form of propositional resizing, Coquand et al. (2017) 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 (2019). Another interesting recent development is a normalization result for cubical type theory proved by Sterling and Angiuli (2021).

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 $\pi_4(\mathbb{S}^3) = \mathbb{Z}/n\mathbb{Z}$ (Brunerie, 2016). This construction has been formalized `Cubical Agda`,[8] 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 (Shulman, 2019). 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 (Riehl, 2019). 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.

Acknowledgements. The author is very grateful to Carlo Angiuli, Elisabeth Bonnevier and Evan Cavallo for commenting on earlier versions of these notes. Multiple explanations and diagrams are borrowed from the Ph.D. thesis of Angiuli (2019). 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. The insightful comments and suggestions by the reviewers and editor also helped substantially improving these notes. The material in this paper is based upon research supported by the Swedish Research Council (SRC, Vetenskapsrådet) under Grant No. 2019-04545.

Notes

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 the "homotopy propositions" or "-1-types" of HoTT/UF. That is, types where all elements are equal.

3 The symbol "よ" is hiragana for "yo".

4 This is not true for simplicial sets where the product of representables has to be subdivided in order to form a simplicial set again. The problem boils down to simplices not being closed under products, for example, $\mathbb{I} \times \mathbb{I}$ is a square which has to be subdivided into two triangles glued together along the diagonal in order to be realized as simplices.

5 Note that we now also use the letter Γ for the cubical set in $\widehat{\square}_{\mathsf{DM}}$ corresponding to the semantic context.

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

7 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.

8 See https://github.com/agda/cubical/blob/ef62b84397396d48135d73ba7400b71c721ddc94/Cubical/Experiments/Brunerie.agda.

References

Angiuli, C. (2019). *Computational Semantics of Cartesian Cubical Type Theory*. Phd thesis, Carnegie Mellon University.

Angiuli, C., Brunerie, G., Coquand, T., Hou (Favonia), K.-B., Harper, R. and Licata, D. R. (2021a). Syntax and Models of Cartesian Cubical Type Theory. Preprint.

Angiuli, C., Cavallo, E., Hou (Favonia), K.-B., Harper, R. and Sterling, J. (2018a). The RedPRL proof assistant (invited paper). In: Blanqui, F. and Reis, G. (eds.), *13th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2018)*, Electronic Proceedings in Theoretical Computer Science, vol. 274.

Angiuli, C., Cavallo, E., Mörtberg, A. and Zeuner, M. (2021b). Internalizing representation independence with univalence. *Proceedings of the ACM on Programming Languages* **5** (POPL). 1–30. https://dl.acm.org/doi/10.1145/3434293

Angiuli, C., (Favonia), K.-B. H. and Harper, R. (2018b). 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*, 6:1–6:17.

Awodey, S. (2018). A cubical model of homotopy type theory. *Annals of Pure and Applied Logic* **169** (12) 1270–1294. Logic Colloquium 2015.

Bezem, M., Coquand, T. and Huber, S. (2014). A model of type theory in cubical sets. In: *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 26, 107–128.

Bezem, M., Coquand, T. and Huber, S. (2019). The univalence axiom in cubical sets. *Journal of Automated Reasoning* **63** 159–171. https://link.springer.com/article/10.1007/s10817-018-9472-6

Bezem, M., Coquand, T. and Parmann, E. (2015). Non-constructivity in Kan simplicial sets. In: Altenkirch, T. (ed.) *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*, Leibniz International Proceedings in Informatics (LIPIcs), vol. 38, Dagstuhl, Germany, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 92–106.

Birkedal, L., Bizjak, A., Clouston, R., Grathwohl, H. B., Spitters, B. and Vezzosi, A. (2019). Guarded cubical type theory. *Journal of Automated Reasoning* **63** 211–253. https://link.springer.com/article/10.1007/s10817-018-9471-7

Brunerie, G. (2016). *On the Homotopy Groups of Spheres in Homotopy Type Theory*. Phd thesis, Université de Nice.

Buchholtz, U. and Morehouse, E. (2017). Varieties of cubical sets. In: Höfner, P., Pous, D. and Struth, G. (eds.) *Relational and Algebraic Methods in Computer Science*, Springer International Publishing, 77–92.

Cavallo, E. and Harper, R. (2019). Higher inductive types in cubical computational type theory. *Proceedings of the ACM on Programming Languages* **3** (POPL) 1:1–1:27.

Cavallo, E., Mörtberg, A. and Swan, A. W. (2020). Unifying cubical models of univalent type theory. In: Fernández, M. and Muscholl, A. (eds.) *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*, Leibniz International Proceedings in Informatics (LIPIcs), vol. 152, Dagstuhl, Germany, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 14:1–14:17.

Clairambault, P. and Dybjer, P. (2011). The biequivalence of locally cartesian closed categories and martin-löf type theories. In: Ong, L. (ed.) *Typed Lambda Calculi and Applications*, Berlin, Heidelberg, Springer, 91–106.

Cohen, C., Coquand, T., Huber, S. and Mörtberg, A. (2018). Cubical type theory: A constructive interpretation of the univalence axiom. In: *Types for Proofs and Programs (TYPES 2015)*, LIPIcs, vol. 69, 5:1–5:34.

Constable, R. L., Allen, S. F., Bromley, H. M., Cleaveland, W. R., Cremer, J. F., Harper, R. W., Howe, D. J., Knoblock, T. B., Mendler, N. P., Panangaden, P., Sasaki, J. T. and Smith, S. F. (1985). *Implementing Mathematics with the Nuprl Proof Development Environment*, Prentice-Hall.

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

Coquand, T. (2015). A cubical type theory. Slides of a talk in Nijmegen, Netherlands.
Coquand, T., Huber, S. and Mörtberg, A. (2018). 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, 255–264.
Coquand, T., Mannaa, B. and Ruch, F. (2017). Stack semantics of type theory. In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 1–11.
Curien, P.-L. (1993). Substitution up to isomorphism. *Fundamenta Informaticae* **19** (1–2) 51–85.
Curien, P.-L., Garner, R. and Hofmann, M. (2014). Revisiting the categorical interpretation of dependent type theory. *Theoretical Computer Science* **546** 99–119.
Dijkstra, E. W. (1974). On the role of scientific thought. http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD447.PDF.
Dybjer, P. (1996). Internal type theory. In: *Lecture Notes in Computer Science*, Berlin, Heidelberg, New York, Springer Verlag, 120–134.
Forsberg, F. N., Xu, C. and Ghani, N. (2020). 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 York, NY, USA, Association for Computing Machinery, 172–185.
Gambino, N. and Sattler, C. (2017). The Frobenius condition, right properness, and uniform fibrations. *Journal of Pure and Applied Algebra* **221** (12) 3027–3068.
Hofmann, M. (1994). On the interpretation of type theory in locally cartesian closed categories. In: *Proceedings of Computer Science Logic*, Lecture Notes in Computer Science, Springer, 427–441.
Hofmann, M. (1997). Syntax and Semantics of Dependent Types. In: A. Pitts & P. Dybjer (Eds.), "*Semantics and Logics of Computation" (Publications of the Newton Institute)*. Cambridge: Cambridge University Press, 79–130. https://www.cambridge.org/core/books/abs/semantics-and-logics-of-computation/syntax-and-semantics-of-dependent-types/119C8085C6A1A0CD7F24928EF866748F
Hofmann, M. and Streicher, T. (1997). Lifting Grothendieck universes. Unpublished Note. Available at https://www2.mathematik.tu-darmstadt.de/ streicher/NOTES/lift.pdf.
Huber, S. (2019). Canonicity for cubical type theory. *Journal of Automated Reasoning* **63** (2) 173–210.
Kapulkin, C. and Lumsdaine, P. L. (2012). The simplicial model of univalent foundations (after Voevodsky). Preprint arXiv:1211.2851v4 [math.LO].
Lambek, J. and Scott, P. J. (1986). *Introduction to Higher Order Categorical Logic*, Cambridge University Press, USA.
Licata, D. R., Orton, I., Pitts, A. M. and Spitters, B. (2018). Internal universes in models of homotopy type theory. In: *FSCD*, LIPIcs, vol. 108. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 22:1–22:17.
Lumsdaine, P. L. and Warren, M. A. (2015). The local universes model: An overlooked coherence construction for dependent type theories. *ACM Transactions on Computational Logic* **16** (3) 1–31.
Martin-Löf, P. (1975). An intiutionistic theory of types: Predicative part. In: Rose, H. E. and Shepherdson, J. (eds.) *Logic Colloquium'73*, Amsterdam, North–Holland, 73–118.
Martin-Löf, P. (1982). Constructive mathematics and computer programming. In: *Logic, Methodology and Philosophy of Science, VI*, 153–175.
Martin-Löf, P. (1984). *Intuitionistic Type Theory*, Bibliopolis.
Martin-Löf, P. (1998). An intuitionistic theory of types. In: *Twenty-Five Years of Constructive Type Theory (Venice, 1995)*, Oxford Logic Guides, vol. 36, New York, Oxford University Press, 127–172.
McBride, C. (2002). Elimination with a motive. In: *Types for Proofs and Programs*, Berlin, Heidelberg, Springer, 197–216.
Mörtberg, A. and Pujet, L. (2020). Cubical synthetic homotopy theory. In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, New York, NY, USA, Association for Computing Machinery, 158–171.
Nordström, B., Petersson, K. and Smith, J. M. (1990). *Programming in Martin-Löf's Type Theory: An Introduction*, USA, Clarendon Press.
Orton, I. and Pitts, A. M. (2018). Axioms for modelling cubical type theory in a topos. *Logical Methods in Computer Science* **14** (4) 1–33.
Parker, J. (2014). Duality between Cubes and Bipointed Sets. Master's thesis, Carnegie Mellon University.
Riehl, E. (2014). *Categorical Homotopy Theory*, Cambridge University Press.
Riehl, E. (2017). *Category Theory in Context*, Dover Publications.
Riehl, E. (2019). The equivariant uniform kan fibration model of cubical homotopy type theory. Talk given at *The International Conference on Homotopy Type Theory (HoTT 2019)* at Carnegie Mellon University.
Sattler, C. (2017). The Equivalence Extension Property and Model Structures. Preprint arXiv:1704.06911v1 [math.CT].
Seely, R. A. G. (1984). Locally cartesian closed categories and type theory. *Mathematical Proceedings of the Cambridge Philosophical Society* **95** (1) 33–48.
Shulman, M. (2019). All $(\infty, 1)$-Toposes have Strict Univalent Universes. Preprint arXiv:1904.07004 [math.AT].
Sterling, J. and Angiuli, C. (2021). Normalization for Cubical Type Theory. Preprint arXiv:2101.11479 [cs.LO].
Swan, A. (2018). Separating Path and Identity Types in Presheaf Models of Univalent Type Theory. Preprint arXiv:1808.00920 [math.LO].

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

Swan, A. and Uemura, T. (2019). On church's thesis in cubical assemblies.
The cubical Development Team (2013). cubical. Available at https://github.com/simhu/cubical/.
The cubicaltt Development Team (2015). cubicaltt. Available at https://github.com/simhu/cubical/.
The mlang Development Team (2019). mlang. Available at https://github.com/molikto/mlang/.
The RedPRL Development Team (2016). RedPRL. Available at http://www.redprl.org/.
The RedPRL Development Team (2018). redtt. Available at https://github.com/RedPRL/redtt.
The RedPRL Development Team (2020). cooltt. Available at https://github.com/RedPRL/cooltt.
The yacctt Development Team (2018). yacctt. Available at https://github.com/mortberg/yacctt/.
Uemura, T. (2019). Cubical assemblies, a univalent and impredicative universe and a failure of propositional resizing. In: Dybjer, P., Santo, J. E. and Pinto, L. (eds.) *24th International Conference on Types for Proofs and Programs (TYPES 2018)*, Leibniz International Proceedings in Informatics (LIPIcs), vol. 130, Dagstuhl, Germany, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 7:1–7:20.
Univalent Foundations Program, T. (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book, Institute for Advanced Study.
Veltri, N. and Vezzosi, A. (2020). Formalizing Pi-calculus in guarded cubical agda. In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, New York, NY, USA, Association for Computing Machinery, 270–283.
Vezzosi, A., Mörtberg, A. and Abel, A. (2019). Cubical agda: A dependently typed programming language with univalence and higher inductive types. *Proceedings of the ACM on Programming Languages* **3** (ICFP) 87:1–87:29.
Vezzosi, A., Mörtberg, A. and Abel, A. (2021). Cubical agda: A dependently typed programming language with univalence and higher inductive types. *Journal of Functional Programming* **31** e8:1–29.
Voevodsky, V. (2009). Notes on type systems. Unpublished note available at https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/expressions_current.pdf (retrieved May, 2021).
Voevodsky, V. (2010). Univalent foundations project. A modified version of an NSF grant application.
Voevodsky, V. (2011). Univalent foundations. Plenary lecture at WoLLIC, May 18.
Voevodsky, V. (2014). The equivalence axiom and univalent models of type theory. (Talk at CMU on February 4, 2010). Preprint arXiv:1402.5556 [math.LO].
Voevodsky, V. (2015). An experimental library of formalized mathematics based on the univalent foundations. *Mathematical Structures in Computer Science* **25** 1278–1294.

Appendix A. Solutions to Exercises

We have collected solutions to the exercises in the notes here. Some results have been formalized in cubicaltt and Cubical Agda in which case we refer to the formalized solutions as well.

Section 3: Cubical type theories and their models

(1) To prove that $\{i\} \times \{j\} \cong \{i, j\}$ in $\square$, we have to analyze what products in $\square$ look like. By the definition of products $\{i\} \times \{j\}$ is an object with morphisms $\pi_1 : \mathrm{Hom}_{\square}(\{i\} \times \{j\}, \{i\})$ and $\pi_2 : \mathrm{Hom}_{\square}(\{i\} \times \{j\}, \{j\})$ satisfying the usual universal property. By the definition of hom-sets in $\square$ we see that π_1 is a function $\{i\} \to (\{i\} \times \{j\}) + 2$ (recall that the functions go in the opposite direction) and similarly for π_2. Furthermore, as the functions go the opposite direction the universal property has to correspond to the universal property of coproducts. Therefore, $\{i\} \times \{j\} = \{i\} \sqcup \{j\} \cong \{i, j\}$.

(2) Given $a, a' : A$ and $b, b' : B$, the binary version of ap can be proved as:

$$\mathsf{ap}_2 : (f : A \to B \to C)\ (p : \mathsf{Path}\ A\ a\ a')\ (q : \mathsf{Path}\ B\ b\ b') \to \mathsf{Path}\ C\ (f\ a\ b)\ (f\ a'\ b')$$
$$\mathsf{ap}_2\ f\ p\ q = \lambda(i : \mathbb{I}).\ f\ (p\ i)\ (q\ i)$$

(3) Given $f, g : A \to B \to C$, the binary (non-dependent) version of funExt can be proved as:

$$\mathsf{funExt}_2 : (p : (x : A)\ (y : B) \to \mathsf{Path}\ C\ (f\ x\ y)\ (g\ x\ y)) \to \mathsf{Path}\ (A \to B \to C)\ f\ g$$
$$\mathsf{funExt}_2\ f\ g\ p = \lambda(i : \mathbb{I})\ (x : A)\ (y : B).\ p\ x\ y\ i$$

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

(4) Negation on Booleans $\mathsf{not} : \mathsf{bool} \to \mathsf{bool}$ can be defined as:

$\mathsf{not} : \mathsf{bool} \to \mathsf{bool}$
$\mathsf{not}\ \mathsf{true} = \mathsf{false}$
$\mathsf{not}\ \mathsf{false} = \mathsf{true}$

We can then prove notK with a helper lemma:

$\mathsf{notK}' : (b : \mathsf{bool}) \to \mathsf{Path}\ \mathsf{bool}\ (\mathsf{not}\ (\mathsf{not}\ b))\ b$
$\mathsf{notK}'\ \mathsf{true} = \mathsf{refl}$
$\mathsf{notK}'\ \mathsf{false} = \mathsf{refl}$

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

(5) Given dependent functions $f, g : (x : A) \to B$, funExt for dependent functions can be defined as:

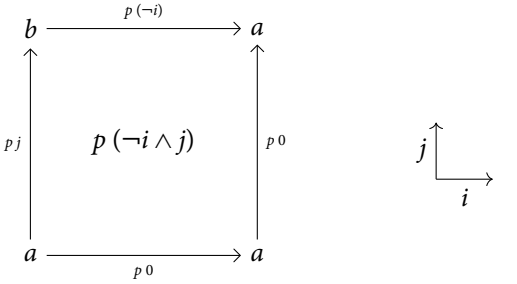
$\mathsf{funExtDep} : (p : (x : A) \to \mathsf{Path}\ B\ (f\ x)\ (g\ x)) \to \mathsf{Path}\ ((x : A) \to B)\ f\ g$
$\mathsf{funExtDep}\ f\ g\ p = \lambda (i : \mathbb{I})\ (x : A).\ p\ x\ i$

Note that the proof is identical to the one of non-dependent funExt. With this, we can give an alternative proof of isPropPi:

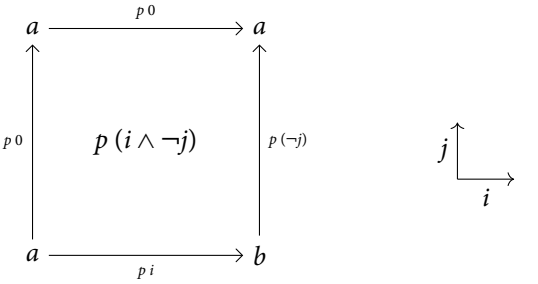
$\mathsf{isPropPi} : (A : \mathscr{U})\ (B : A \to \mathscr{U})\ (h : (x : A) \to \mathsf{isProp}\ (B\ x)) \to \mathsf{isProp}\ ((x : A) \to B)$
$\mathsf{isPropPi}\ A\ B\ h = \lambda\ (f\ g : (x : A) \to B).\ \mathsf{funExtDep}\ (\lambda (x : A).\ h\ x\ (f\ x)\ (g\ x))$

(6) Given $p : \mathsf{Path}\ A\ a\ b$ and $i, j : \mathbb{I}$, we can draw the squares corresponding to the various connections as:

i.

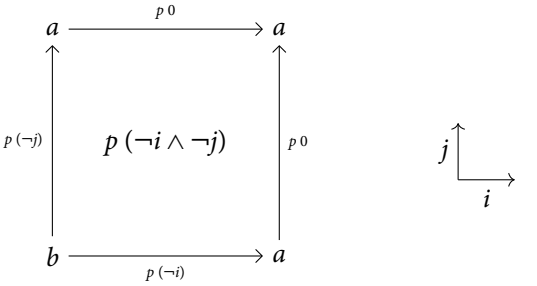


ii.

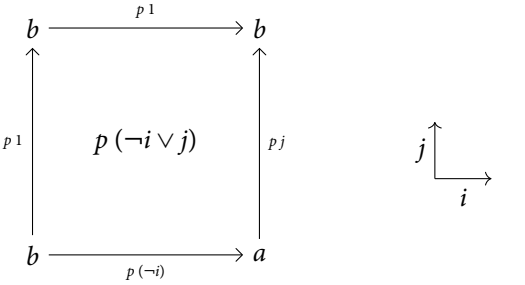


https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

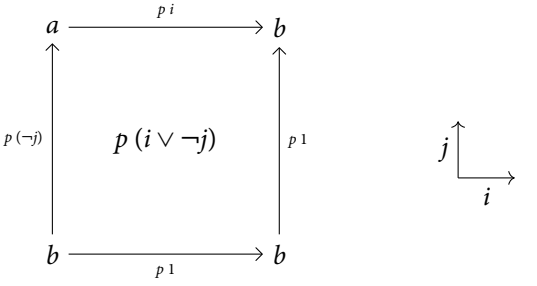
iii.



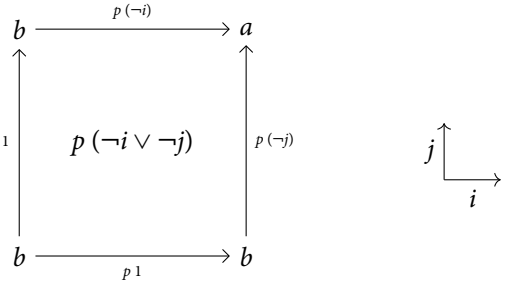
iv.



v.



vi.



(7) The solution to this problem is analogous to the one above, but with cubes instead of squares.

(8) The hom-set $\mathrm{Hom}_{\square}(\{i_1, \ldots, i_n\}, \{i\})$ correspond to the set of functions $\{i\} \to \{i_1, \ldots, i_n\} + \{0, 1\}$, so it has $n + 2$ elements. The hom-set $\mathrm{Hom}_{\square_{\mathsf{DM}}}(\{i_1, \ldots, i_n\}, \{i\})$ on the other hand is the set of functions $\{i\} \to \mathsf{DM}(\{i_1, \ldots, i_n\})$. This is the number of elements of the free distributive lattice on $2n$ generators as negations can be pushed down to the names (so the

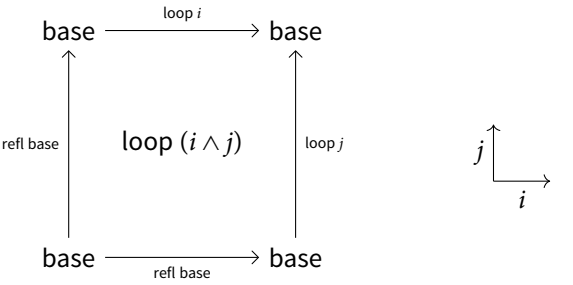
https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

generators are $\{i_1, \ldots, i_n, \neg i_1, \ldots, \neg i_n\}$). These numbers are called the "Dedekind numbers" and grow incredibly fast (the 10:th Dedekind number is in fact not even known). See https://en.wikipedia.org/wiki/Dedekind_number and https://oeis.org/A000372 for details.

(9) Given $A : \mathscr{U}$, the alternative version of contrSingl can be proved as:

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

(10) The full square looks like this:



So loop $(i \wedge j)$ is an identification of refl base and loop *over* an identification of refl base and loop. This square does hence not prove that loop is trivial; for this to be the case, all but one side would have had to been refl base.

(11) The homotopy can be constructed as:

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

(12) Assuming decidable equality on $\mathbb{I}$ internally means that we have a function:

$$\mathsf{dec} : \Pi(i\ j : \mathbb{I}).\ \mathsf{bool}$$

that returns true if $i \equiv j$ and false otherwise. With this we can define

$$\mathsf{oops} : \mathsf{Path}_{\mathscr{U}}(\mathsf{Unit}, \mathsf{Empty})$$
$$\mathsf{oops} = \lambda(i : \mathbb{I}).\ \mathsf{if}\ \mathsf{dec}\ i\ 0\ \mathsf{then}\ \mathsf{Unit}\ \mathsf{else}\ \mathsf{Empty}$$

Section 4: Cubical transport

(1) In order to give the *negative* definition of cubical transport for $A \times B$, we first have to consider $\mathsf{transport}^i\ (A \times B)\ p$ to be a neutral value and give the reduction rules when applying the eliminators to this. This is easily done by adding the equations:

$$(\mathsf{transport}^i\ (A \times B)\ p).1 = \mathsf{transport}^i\ A\ p.1$$
$$(\mathsf{transport}^i\ (A \times B)\ p).2 = \mathsf{transport}^i\ B\ p.2$$

(2) To define cubical transport in sum types $A + B$, we assume that we are given $x : A(0/i)$ and $y : B(0/i)$, and write:

$$\mathsf{transport}^i\ (A + B)\ (\mathsf{inl}\ x) = \mathsf{inl}\ (\mathsf{transport}^i\ A\ x)$$
$$\mathsf{transport}^i\ (A + B)\ (\mathsf{inr}\ y) = \mathsf{inr}\ (\mathsf{transport}^i\ B\ y)$$

(3) When defining cubical transport for Σ- and Π-types, one runs into the problem that one needs a path connecting an element with its transport (just like was needed to prove the computation rule for J). Indeed, consider the case of cubical transport in Σ-types. For this,

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

we need to fill the ? in:

$$\frac{\Gamma, i : \mathbb{I} \vdash (x : A) \times B : \mathscr{U} \qquad \Gamma \vdash p : (x : A(0/i)) \times B(0/i)}{\Gamma \vdash \mathsf{transport}^i\ ((x : A) \times B)\ p = ?}$$

To this end we can write down the following:

$$(\mathsf{transport}^i\ A\ p.1, \mathsf{transport}^i\ (B\ ?)\ p.2)$$

There is however no way to fill this ? at this point as we would need a path from $p.1$ to $\mathsf{transport}^i\ A\ p.1$. Indeed, $p.2$ is in $B(0/i)\ p.1$, but the result should be in $B\ (1/i)\ (\mathsf{transport}^i\ A\ p.1)$ as we have picked $\mathsf{transport}^i\ A\ p.1$ as the first component of the result pair. Note that there is not really anything else we could have picked for the first component as we need something in $A(1/i)$ and the only way to get something of this type with what we're given is by transporting $p.1$ in A.

The problem when defining cubical transport for Π-types is similar, but there one needs to also handle the reversal coming from transporting backwards.

Section 5: Kan composition operations

(1) i.

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

ii.

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

iii.

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

(2) This is quite a bit more difficult and the interested reader can consult: https://github.com/agda/cubical/blob/ef62b84397396d48135d73ba7400b71c721ddc94/Cubical/Foundations/GroupoidLaws.agda#L84.

(3) i. For a proof in `cubicaltt` see: https://github.com/mortberg/cubicaltt/blob/a5c6f94bfc0da84e214641e0b87aa9649ea114ea/examples/prelude.ctt#L165 For a proof in `Cubical Agda` of a special case which appears in the Hopf fibration see: https://github.com/agda/cubical/blob/ef62b84397396d48135d73ba7400b71c721ddc94/Cubical/HITs/S1/Base.agda#L392

ii. This is a direct adaptation of the above `cubicaltt` proof.

iii. These squares would prove that $p \cdot p = q \cdot q$ and $q \cdot p = p \cdot q$. This is not always true in the presence of univalence. For example if p identifies $\mathbb{Z}$ with itself by adding 1 and q just flips the sign. Then transporting along $p \cdot p$ will add 2, but transporting along $q \cdot q$ will just flip the sign twice. Also, transporting along $q \cdot p$ will flip the sign and add 1, so -3 will be mapped to 4. But transporting along $p \cdot q$ will add 1 and then flip the sign, so -3 will be mapped to 2. These paths hence cannot be equal in general in the presence of univalence.

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

iv. This is substantially harder and various solution can be found in: https://github.com/mortberg/cubicaltt/blob/a5c6f94bfc0da84e214641e0b87aa9649ea114ea/examples/constcubes.ctt.

(4)

$$\mathsf{isContrProp} : \mathsf{isContr}\ A \to \mathsf{isProp}\ A$$
$$\mathsf{isContrProp}\ h\ x\ y = \lambda(i : \mathbb{I}).\ \mathsf{comp}^j\ A\ [\ (i = 0) \mapsto h.2\ x\ (\neg j), (i = 1) \mapsto h.2\ y\ (\neg j)\ ]\ h.1$$

(5) For a `Cubical Agda` proof see: https://github.com/agda/cubical/blob/ef62b84397396d48135d73ba7400b71c721ddc94/Cubical/Foundations/Prelude.agda#L406 (note that hcomp is just comp^i where the type line doesn't depend on i).

(6) For a `Cubical Agda` proof see: https://github.com/agda/cubical/blob/ef62b84397396d48135d73ba7400b71c721ddc94/Cubical/Foundations/Prelude.agda#L418.

(7)

$$\mathsf{isPropRetract} : (h : (x\ :\ A) \to \mathsf{Path}\ A\ (g\ (f\ x))\ x) \to \mathsf{isProp}\ B \to \mathsf{isProp}\ A$$
$$\mathsf{isPropRetract}\ h\ p\ x\ y =$$
$$\lambda(i : \mathbb{I}).\ \mathsf{comp}^j\ A\ [\ (i = 0) \mapsto h\ x\ j, (i = 1) \mapsto h\ y\ j\ ]\ (g\ (p\ (f\ x)\ (f\ y)\ i))$$

Section 6: Glue types and univalence

(1) We can prove that the identity function is an equivalence using a connection as

$$\mathsf{idfun}_A : A \to A$$
$$\mathsf{idfun}_A\ x = x$$

$$\mathsf{id}_A : \mathsf{Equiv}\ A\ A$$
$$\mathsf{id}_A = (\mathsf{idfun}_A, \lambda(y : A).\ ((y, \mathsf{refl}), \lambda(z : \mathsf{Fiber}\ \mathsf{idfun}_A\ y)\ (i : \mathbb{I}).\ (z.2\ (\neg i), \lambda(j : \mathbb{I}).\ z.2\ (\neg i \vee j))))$$

(2) Proving isoToEquiv is quite involved and a proof in `cubicaltt` can be found at: https://github.com/mortberg/cubicaltt/blob/a5c6f94bfc0da84e214641e0b87aa9649ea114ea/examples/equiv.ctt#L221
A proof in `Cubical Agda` can be found at: https://github.com/agda/cubical/blob/ef62b84397396d48135d73ba7400b71c721ddc94/Cubical/Foundations/Isomorphism.agda#L104.

(3) Using the solution of exercise (4) in Section 3, it is easy to prove that not is an equivalence:

$$\mathsf{notEquiv} : \mathsf{Equiv}\ \mathsf{bool}\ \mathsf{bool}$$
$$\mathsf{notEquiv} = \mathsf{isoToEquiv}\ \mathsf{not}\ \mathsf{not}\ \mathsf{notK}'\ \mathsf{notK}'$$

Using ua we can turn this into a path:

$$\mathsf{notPath} : \mathsf{Path}\ \mathscr{U}\ \mathsf{bool}\ \mathsf{bool}$$
$$\mathsf{notPath} = \mathsf{ua}\ \mathsf{notEquiv}$$

By ua_β, we know that transporting true along this path will apply the function underlying the equivalence, so the result will be false. Note that this will compute judgmentally despite ua_β only holding up to a path. The reason being that $\mathsf{transport}^i\ \mathsf{bool}\ b$ is just b as bool has no parameters.

https://doi.org/10.1017/S0960129521000311 Published online by Cambridge University Press

(4)

$$\mathsf{uaIdEquiv} : \mathsf{Path}\ (\mathsf{Path}\ \mathscr{U}\ A\ A)\ (\mathsf{ua}\ \mathsf{id}_A)\ \mathsf{refl}$$
$$\mathsf{uaIdEquiv}\ i\ j = \mathsf{Glue}\ [(i = 1) \vee (j = 0) \vee (j = 1) \mapsto (A, \mathsf{id}_A)]\ A$$

A formalized proof of this can be found at: https://github.com/agda/cubical/blob/ef62b84397396d48135d73ba7400b71c721ddc94/Cubical/Foundations/Univalence.agda#L40

Cite this article: Mörtberg A (2021). Cubical methods in homotopy type theory and univalent foundations. *Mathematical Structures in Computer Science* **31**, 1147–1184. https://doi.org/10.1017/S0960129521000311