Meaning explanations at higher dimension

Carlo Angiuli   Robert Harper
Carnegie Mellon University

November 2016

Abstract

Martin-Löf's intuitionistic type theory is a widely-used framework for constructive mathematics and computer programming. In its most popular form, type theory consists of a collection of inference rules inductively defining formal proofs. These rules are justified by Martin-Löf's meaning explanations, which extend the Brouwer-Heyting-Kolmogorov interpretation of connectives to a rich collection of types, and therefore provide a constructive realizability interpretation of formal proofs.

Around 2005, researchers noticed that the rules of type theory also admit homotopy-theoretic models, and subsequently extended type theory with constructs inspired by these models: higher inductive types and Voevodsky's univalence axiom. Although the resulting homotopy type theory has proved useful for homotopy-theoretic reasoning, it lacks a constructive interpretation. In this overview, we discuss a cubical generalization of the meaning explanations of type theory that constitutes an inherently constructive account of higher-dimensional structure in types.

1 Introduction

Martin-Löf's intuitionistic type theory is a comprehensive theory of constructions intended as a framework for constructive mathematics and computer programming [Martin-Löf, 1975, 1982, 1984]. There are two rather different traditions of type theory, both due to Martin-Löf, embodying two distinct modes of use of constructive logic.

Its most widely-used form, which we call *formal type theory*, is given by a collection of inference rules defining a collection of types and their inhabitants, and specifying when two such are to be considered definitionally equal [Martin-Löf, 1975, 1984; The Univalent Foundations Program, 2013]. Formal type theory serves as an axiomatic logical framework for constructive mathematics; it is especially well-suited to the mechanization of mathematics, as shown by theorem provers like Coq [The Coq Project, 2016], Agda [Norell, 2007], and Lean [de Moura et al., 2015]. Formal type theory embraces *axiomatic freedom* by neither admitting nor contradicting non-constructive principles such as the unrestricted law of the excluded middle [Bishop, 1967]; this freedom allows many interesting interpretations, most notably variants of the well-known proofs-as-programs [Howard, 1980] and realizability [Kleene, 1952] interpretations.

Computational type theory, in contrast, is an account of mathematics based on the concept of a program or algorithm [Martin-Löf, 1982; Constable, et al., 1985]. Types and their inhabitants are computer programs satisfying an open-ended collection of criteria known as *meaning explanations*, which extend the Brouwer-Heyting-Kolmogorov interpretation of the connectives of intuitionistic logic [Troelstra and van Dalen, 1988; Sundholm, 1983]. By its open-ended nature, computational type theory supports many forms of constructions, including general recursion [Constable and Smith, 1993] and Brouwer's notion of bar recursion [Rahli and Bickford, 2015]. The authors find computational type theory appealing for its direct grounding in computer science—types are inhabited, quite literally, by programs constructing the desired objects.

A distinctive feature of certain formal type theories is the *identity type* $a =_A b$ of proofs that $a : A$ and $b : A$ are equal, defined by the rules [Martin-Löf, 1975; The Univalent Foundations Program, 2013]:

$$\frac{\Gamma \vdash A\ \mathsf{type} \quad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{refl}_a : a =_A a} \qquad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : A \quad \Gamma \vdash p' : a =_A b \quad \Gamma, x : A, y : A, p : x =_A y \vdash C\ \mathsf{type} \quad \Gamma, z : A \vdash c : C[z, z, \mathsf{refl}_z/x, y, p]}{\Gamma \vdash \mathsf{ind}_{=_A}(x.y.p.C, z.c, a, b, p') : C[a, b, p'/x, y, p]}$$

The introduction rule establishes that every $a : A$ is identical to itself, and the elimination rule describes what one can conclude from a proof p that $a : A$ and $b : A$ are identical. From these rules one can prove identity is reflexive, symmetric, and transitive.

A natural question is, are two proofs of $a =_A b$ necessarily identical? Hofmann and Streicher [1998] showed that this property, known as *uniqueness of identity proofs*, is not provable in formal type theory, by means of a countermodel in which types are *groupoids* (categories in which all morphisms are invertible), $a : A$ is an object of A, and $p : a =_A b$ is a morphism between a and b in A. Uniqueness of identity proofs fails here because parallel morphisms need not be equal.

Around 2005, a number of researchers noticed that the groupoid model of Hofmann and Streicher can be generalized to homotopy-theoretic models in Quillen model categories [Awodey and Warren, 2009], weak factorization systems [Gambino and Garner, 2008], weak ω-groupoids [Lumsdaine, 2009; van den Berg and Garner, 2011], and simplicial sets [Voevodsky, 2006; Streicher, 2006]. These models suggest that formal type theory can serve as an axiomatic framework for homotopy-theoretic constructions, and in particular, reasoning about topological spaces. In such an interpretation, one thinks of a type as a space, $a : A$ as a point in A, $p : a =_A b$ as a path between a and b in A, and $f : A \to B$ as a continuous map from A to B. These paths are reflexive, symmetric, and transitive, and form a groupoid modulo *paths between paths* $\alpha : p =_{(a=_A b)} q$ witnessing the unitality, cancellation, and associativity of these operations.

However, as formal type theory was not developed with higher-dimensional models in mind, none of its types correspond to interesting spaces. (Nothing in standard type theory contradicts uniqueness of identity proofs.) This can be remedied by adding new homotopically-inspired constructs to type theory: *higher inductive types*, inductive types generated by not only points but also paths [Shulman, 2011]; and Voevodsky's *univalence axiom*, stating that identity of types is homotopy equivalence (a higher-dimensional analogue of isomorphism) [Voevodsky, 2010; Kapulkin and Lumsdaine, 2016]. Formal type theory augmented with axioms for higher inductive types and univalence is considered in *Homotopy Type Theory: Univalent Foundations of Mathematics* [The Univalent Foundations Program, 2013]; we will henceforth call this theory *formal homotopy type theory (HoTT)*.

From a mathematical perspective, formal HoTT is a useful axiomatic framework for homotopy theory, and especially the mechanization thereof. Higher inductive types can represent many spaces and constructions on spaces, from n-dimensional spheres to homotopy pushouts. Univalence enables analysis of the path structure of such spaces, and ensures such constructions respect homotopy equivalence (because constructions in type theory respect identity). Many results have already been proved using these tools, ranging from basic category theory to powerful theorems of algebraic topology [The Univalent Foundations Program, 2013; Voevodsky et al.].

Philosophically, however, the haphazard introduction of additional axioms to type theory is troubling. Although the rules of formal type theory are justified through pre-mathematical explanations of the judgments and logical connectives [Martin-Löf, 1984], higher inductive types and univalence are justified through appeals to mathematical models defined with classical logic. In particular, the proofs-as-programs interpretations of formal type theory do not obviously extend to formal HoTT.

This defect has real consequences for practitioners of formal HoTT. In 2013, Brunerie proved that a particular topological invariant is given by the group $\mathbb{Z}/n\mathbb{Z}$ where $\cdot \vdash n : \mathbb{N}$ [Brunerie, 2013]. A proofs-as-programs interpretation for formal HoTT would provide a straightforward method for computing the numeral to which n is definitionally equal. Instead, n is not definitionally equal to a numeral, and Brunerie was not able to prove that this invariant is $\mathbb{Z}/2\mathbb{Z}$ until 2016, after developing a second, more elaborate proof [Brunerie, 2016]. If the existence of a natural number does not require the ability to exhibit a numeral, it is not clear in what sense formal HoTT is a constructive theory!

In this overview paper, we discuss an extension of the meaning explanations of computational type theory that properly accounts for higher-dimensional structure in types. This extension is based on two key insights: that the judgmental apparatus of type theory must itself be generalized to higher dimension [Licata and Harper, 2012], and that *abstract cubes* afford a convenient syntactic representation of higher-dimensional structure [Bezem et al., 2014].

The resulting *cubical meaning explanations* can be used as the basis of a computational higher-dimensional type theory [Angiuli et al., 2017]; they also serve as cubical realizability models of formal higher-dimensional type theories [Licata and Harper, 2012; Licata and Brunerie, 2014; Cohen et al., 2016]. Philosophically, they explain the higher-dimensional content of logical connectives, and hence represent an extension of the Brouwer-Heyting-Kolmogorov interpretation to higher dimension.

2 Meaning explanations

We begin by recalling Martin-Löf's meaning explanations of computational type theory [Martin-Löf, 1982; Constable, et al., 1985; Allen, 1987a,b]. Type theory is built on *judgments*, forms of assertion that are conceptually prior to the concepts of type or membership. The four basic judgments express typehood (and equality of types) and membership (and equality of members):

$$\begin{array}{ll} A\ \mathsf{type} & A \doteq B\ \mathsf{type} \\ M \in A & M \doteq N \in A \end{array}$$

These judgments range over an *open-ended* notion of program, in the sense that certain programs are postulated to be meaningful, but nothing relies on the nonexistence of certain programs—there is no extremal clause stating that those given are all and only the programs in question.[1] In fact, one can even consider classical set-theoretic functions as programs [Howe and Stoller, 1994].

A programming language is specified by defining the syntax of expressions, including concepts of binding, scope, and substitution for variables, using methods codified in the theories of *arities* [Nordström et al., 1990] and *abstract binding trees* [Harper, 2016]. Expressions are given computational meaning by specifying which are *canonical*, i.e., not subject to further computation; and by defining an *operational semantics* that deterministically simplifies closed expressions in a process that may or may not terminate with an expression in canonical form. Any fully expressive computation system must admit nontermination, and type theories differ to the extent that nonterminating expressions participate meaningfully in types and members [Constable and Smith, 1988].

We first specify the meanings of the basic judgments $A\ \mathsf{type}$ and $M \in A$, where M and A are both closed programs, in terms of their computational behavior. The former states that the program A evaluates to a canonical type A_0, meaning that we know what are the canonical members of A_0 and when two such are equal. (Logically, this corresponds to knowing that A is a proposition, because we know what counts as evidence for A.) The latter states that the program M evaluates to a canonical member of the canonical type given by A. (Logically: M computes evidence for the truth of the proposition A.) The canonical types and their canonical members are defined on a case-by-case basis according to their outermost form.

[1] The NuPRL computational type theory adopts a mild constraint on possible programs introduced by Howe [1989], in the interest of increasing the utility of the theory. To date, no proposed extension of the theory has run afoul of this constraint.

Closely related are the judgments $A \doteq B$ type and $M \doteq N \in A$; the former states that A and B evaluate to equal canonical types (whose canonical members are the same), and the latter states that M and N evaluate to equal canonical members of the canonical type given by A. Equality of members depends on the type at which they are considered, so it must be defined explicitly for each type. For example, the identity function and the absolute value function are equal as members of the type of functions on the natural numbers, but are of course distinct functions over the (positive and negative) integers. In fact, as a matter of technical convenience, one can define only the binary forms of judgment, and recover A type and $M \in A$ as reflexive instances—to be a type is to be an equal type to oneself, and similarly for members of a type.

We then define the *open judgments*:

$$a_1 : A_1, \ldots, a_n : A_n \gg A \doteq B \ \mathsf{type}$$
$$a_1 : A_1, \ldots, a_n : A_n \gg M \doteq N \in A$$

for open expressions M, N, A, B. These are defined by induction on the number $n \geq 0$ of free variables by means of *functionality*, i.e., type equality and member equality must respect equality of closed instances in each variable. For instance, $a_1 : A_1 \gg A \doteq B$ type states that for any $M \doteq N \in A_1$, we know $A[M/a_1] \doteq B[N/a_1]$ type. Open terms are thus regarded extensionally as maps sending equal members of A_i to equal members of instances of A.

Finally, the role of types is to internalize mathematical statements about judgments. For instance, a canonical member of the product type $A \times B$ is a pair $\langle M, N \rangle$ of terms such that $M \in A$ and $N \in B$; two canonical members $\langle M, N \rangle$ and $\langle M', N' \rangle$ are equal when $M \doteq M' \in A$ and $N \doteq N' \in B$. That is, (equal) evidence for the conjunction of propositions is a pair of (equal) evidence for each proposition: it is in this sense that the meaning explanations are an extensional form of the Brouwer-Heyting-Kolmogorov interpretation. Similarly, equal canonical members of the function type $A \to B$ are maps $\lambda a.M$ and $\lambda a.M'$ such that $a : A \gg M \doteq M' \in B$, and refl_M is a canonical member of the identity type $M =_A N$ whenever $M \doteq N \in A$. Such definitions proceed in a predicative fashion, in which successive definitions build on prior ones.

The meaning explanations of type theory *define* the truth of propositions in terms of constructions witnessing them—they are not merely a (re)interpretation of proofs as programs, but rather a guarantee of the inherently constructive nature of mathematics performed in computational type theory. Note, however, that proving a proposition requires not only exhibiting a construction, but also recognizing that this construction produces the desired object. Such recognition is no trivial matter: one cannot always mechanically determine whether a program terminates, much less the form of its answer!

For this reason, especially when mechanizing proofs, it is useful to isolate a manageable fragment of computational type theory to serve as a window on the full truth. Formal type theory is one such fragment, and constitutes an inductively-defined *proof theory* for the meaning explanations. We notate its judgments $\Gamma \vdash A\ \mathsf{type}$ and $\Gamma \vdash M : A$, to distinguish them from the meaning explanations described above. The latter asserts that M is a formal derivation of the proposition A; in particular, it is decidable whether M indeed proves A, with no further information. Thus formal type theory is directly suitable for implementation in a theorem prover.

Thus, although formal type theory is not defined by the meaning explanations, it is in a strong sense motivated by them. For example, because formal proofs are not inherently programs, establishing a proofs-as-programs interpretation requires metamathematics; on the other hand, this interpretation should, morally, be inherited from the meaning explanations.

Adding rules of proof construction without accounting for them in the meaning explanations therefore destroys our justification of *all* rules. In the authors' opinion, metamathematical concerns about formal HoTT—such as the lack of a proofs-as-programs interpretation, or the failure of the *canonicity property* that every term $\cdot \vdash n : \mathbb{N}$ is definitionally equal to a numeral—are merely symptoms of this deeper problem, which can only be addressed by developing meaning explanations that support higher inductive types and univalence by directly accounting for computation at higher dimension.

3 Abstract cubes

Formal HoTT disrupts the standard meaning explanations by adding non-refl constants to various identity types: at higher inductive types, for path constructors; and at the type universe, for each equivalence between types. A simple instance of this phenomenon occurs for the *interval* higher inductive type $\mathbb{I}$, defined in part by the rules:

$$\frac{}{\Gamma \vdash \mathsf{zero} : \mathbb{I}} \qquad \frac{}{\Gamma \vdash \mathsf{one} : \mathbb{I}} \qquad \frac{}{\Gamma \vdash \mathsf{seg} : \mathsf{zero} =_{\mathbb{I}} \mathsf{one}}$$

Pictorially, $\mathbb{I}$ is generated by two points and a path connecting them:

$$\mathsf{zero} \xrightarrow{\mathsf{seg}} \mathsf{one}$$

The point constructors zero and one are distinct canonical elements of $\mathbb{I}$, and seg is a canonical path of $\mathbb{I}$ from zero to one. One can see that the identity type of formal HoTT does not internalize equality, as zero and one are not identical.

There are, however, additional paths in $\mathbb{I}$ beyond seg itself. All elements of the identity type are invertible, so it follows (by the elimination rule of the identity type) that $\mathbb{I}$ also has a path $\mathsf{seg}^{-1} : \mathsf{one} =_{\mathbb{I}} \mathsf{zero}$. This path, too, has an inverse $(\mathsf{seg}^{-1})^{-1} : \mathsf{zero} =_{\mathbb{I}} \mathsf{one}$, which is not identical to the original seg (although they must be connected by a path). None of these paths have a simpler description; they are all, like seg, canonical paths of $\mathbb{I}$.

Following Martin-Löf's methodology that types internalize judgments, a proper accounting of identity-as-paths requires extending the membership judgment to account for path membership. The 2-dimensional formal type theory of Licata and Harper [2012] was the first type theory to include judgments capturing not only typehood and membership, but also that p is a path between a and b in A (and that p, q are equal such paths):

$$\begin{array}{cc} \Gamma \vdash A \ \mathsf{type} & \Gamma \vdash A \equiv B \ \mathsf{type} \\ \Gamma \vdash a : A & \Gamma \vdash a \equiv b : A \\ \Gamma \vdash p : a \simeq_A b & \Gamma \vdash p \equiv q : a \simeq_A b \end{array}$$

The path judgment has rules stating that paths are composable and invertible:

$$\frac{\Gamma \vdash p : a \simeq_A b \quad \Gamma \vdash q : b \simeq_A c}{\Gamma \vdash q \circ p : a \simeq_A c} \qquad \frac{\Gamma \vdash p : a \simeq_A b}{\Gamma \vdash p^{-1} : b \simeq_A a}$$

and the only role of the identity type $a =_A b$ is to internalize the path judgment:

$$\frac{\Gamma \vdash p : a \simeq_A b}{\Gamma \vdash \mathsf{in}\ p : a =_A b} \qquad \frac{\Gamma \vdash q : a =_A b}{\Gamma \vdash \mathsf{out}\ q : a \simeq_A b}$$

Because this type no longer internalizes identity, we will henceforth call it (and future extensions) the *identification* or *path type*.

Licata and Harper [2012] establish canonicity for their formal type theory, which includes also an instance of the univalence axiom, namely, a non-trivial path between bool and itself corresponding to the equivalence swapping its points. Canonicity follows by an adaptation of the Hofmann and Streicher [1998] groupoid model, in which A is a groupoid, $a : A$ is an object of A, and $p : a \simeq_A b$ is a morphism between a and b in A.

The shortcoming of Licata and Harper [2012] is that its judgmental structure does not account for structure above the second dimension—paths between paths, paths between those paths, and so forth. Indeed, there are no non-trivial paths between paths in the theory. To be sure, one could extend this theory with a judgment that α is a path between paths p and q between a and b in A:

$$\Gamma \vdash \alpha : p \simeq_{a \simeq_A b} q$$

and stipulate that paths between paths are composable and invertible, and that there are paths between paths witnessing the unitality, cancellation, and associativity of composition and inversion of one-dimensional paths. But what of the next dimension, in which γ is a path between paths α and β between paths p and q between a and b in A... it is not obvious how to define an infinite tower of judgments, much less the infinitely many operations they must support!

The solution, as observed by Bezem et al. [2014], is to instead represent this iterated path structure *cubically*. That is, we consider not the points, paths, and paths between paths of a type, but instead the points, paths, and *squares*:

$$
a \qquad\qquad a \xrightarrow{p} b \qquad\qquad \begin{tikzcd} a \arrow[r, bend left, "p"] \arrow[r, bend right, "q"'] \arrow[r, phantom, "\alpha"] & b \end{tikzcd}
$$

$$
a \qquad\qquad a \xrightarrow{p} b \qquad\qquad \begin{tikzcd} a \arrow[r, "p"] \arrow[d, "q"'] \arrow[dr, phantom, "\alpha"] & b \arrow[d, "s"] \\ c \arrow[r, "r"'] & d \end{tikzcd}
$$

n-cubes are parametrized by n *dimension* variables $x, y, z, \ldots$, which can be thought of as ranging over an interval. For instance, a 2-cube (square) M varies over two dimensions, say x and y. Instantiating x to 0 yields a 1-cube (line) $M\langle 0/x\rangle$ varying only over y. One can further instantiate the y to 1, obtaining a 0-cube (point) $M\langle 0/x\rangle\langle 1/y\rangle$. This process selects first the left edge of M, followed by the bottom end point of that line:

$$
\begin{tikzcd}
M\langle 0/x\rangle\langle 0/y\rangle \arrow[rr, "M\langle 0/y\rangle"] \arrow[dd, "M\langle 0/x\rangle"'] & & M\langle 1/x\rangle\langle 0/y\rangle \arrow[dd, "M\langle 1/x\rangle"] \\
& M & \\
M\langle 0/x\rangle\langle 1/y\rangle \arrow[rr, "M\langle 1/y\rangle"'] & & M\langle 1/x\rangle\langle 1/y\rangle
\end{tikzcd}
$$

On the other hand, one could select the left end point of the bottom edge by instantiating y to 1 and then x to 0; the geometric fact that these points coincide follows from the syntactic fact that instantiations of distinct variables commute.

Cubes afford us a uniform description of n-dimensional members of types as terms parametrized over n dimension variables, rather than as n-fold iterations of paths. The order of these dimension variables is immaterial, allowing us to think of a square either as an identification along x of two y-lines, or as an identification along y of two x-lines. By fixing how one is permitted to instantiate dimension variables, we can specify the allowed geometric operations:

1. $\langle 0/x\rangle$ and $\langle 1/x\rangle$, as discussed above, compute the *faces* of a cube.

2. $\langle x/y\rangle$ computes the *diagonal* of a cube, so that the variation in x traces out a simultaneous variation in both the x and y directions. In the diagram above, $M\langle x/y\rangle$ is the upper-left-to-lower-right diagonal of M.

3. We can also *degenerate* a cube by regarding it as parametrized by an additional dimension z which happens not to be used. For example, the square M is a cube in x, y, z constant in the z dimension.

One must also describe how faces, diagonals, and degeneracies interact: for example, taking a z-face of a cube degenerated in z yields the original cube.

These operations and laws yield a *structural* or *cartesian* notion of cube. Further operations are also possible:

4. $\langle (1-x)/x \rangle$ computes the *reversal* of a cube, whose faces in the x direction are now swapped.

5. $\langle (x \wedge y)/x \rangle$ and $\langle (x \vee y)/x \rangle$ compute the *connections* of a cube, for instance, turning an x-line M into an (x, y)-square whose bottom and right (resp., top and left) faces are M.

Cubes as a device for representing spaces are a storied mathematical idea due to Kan [1955]. Bezem et al. [2014] were the first to use cubes in the context of type theory, building a constructive model of formal type theory in cubical sets (with faces and degeneracies). Following this idea, there is ongoing research on a variety of higher-dimensional type theories with cubical judgments, including a formal type theory with cartesian cubes [Licata and Brunerie, 2014], the authors' work with Todd Wilson on a computational type theory with cartesian cubes [Angiuli et al., 2017], and a formal type theory using cubes equipped with all the operations described above [Cohen et al., 2016].

4 Cubical meaning explanations

We proceed by describing the cubical meaning explanations of computational cubical type theory as developed by Angiuli et al. [2017], which directly generalize Martin-Löf's meaning explanations of ordinary computational type theory. Although some technical details of this presentation are specific to the authors' type theory, we expect similar principles will justify other cubical type theories, including the formal cubical type theories of Licata and Brunerie [2014] and Cohen et al. [2016]. (Indeed, the proof of canonicity for the latter type theory shares some features with our meaning explanations [Huber, 2016].)

The four basic judgments of computational cubical type theory are:

$$\begin{array}{cc} A \ \mathsf{pretype}\ [x_1, \ldots, x_n] & A \doteq B \ \mathsf{pretype}\ [x_1, \ldots, x_n] \\ M \in A\ [x_1, \ldots, x_n] & M \doteq N \in A\ [x_1, \ldots, x_n] \end{array}$$

where $x_1, \ldots, x_n$ are lists of dimension variables. These judgments range over an open-ended notion of *cubical program*, which includes ordinary programs as well as programs containing dimension variables. Cubical programs are given computational meaning by an operational semantics which deterministically simplifies expressions that are closed on term variables (but may contain dimension variables). When this process terminates, it yields an expression in canonical form that, again, may contain dimension variables. The operational semantics, and therefore also the judgments, respect renaming of dimension variables.

A program whose free dimension variables are contained in $x_1, \ldots, x_n$ represents an n-cube. Cubical operations (faces, diagonals, etc.) are computed by substituting dimension expressions for dimension variables: $\langle 0/x_i \rangle$ or $\langle 1/x_i \rangle$ to compute the left (resp., right) face in dimension x_i, or $\langle x_j/x_i \rangle$ to compute the diagonal identifying x_i and x_j.

Consider, for example, the interval higher inductive type $\mathbb{I}$ described in Section 3. Its point constructors are terms zero and one as before; its path constructor, in direction x, is the term seg_x. The terms zero, one, and seg_x are canonical, but seg_0 evaluates to zero, ensuring that the left face operation on seg_x, $\mathsf{seg}_x\langle 0/x \rangle$, indeed computes zero (and similarly for seg_1 and one).

The basic judgment A $\mathsf{pretype}$ $[\Psi]$ implies A evaluates to a canonical Ψ-dimensional pretype A_0, meaning that we know what are the canonical Ψ-dimensional members of A_0 and when two such are equal. The basic judgment $M \in A$ $[\Psi]$ implies that M evaluates to a canonical Ψ-dimensional member of the canonical pretype A_0. The members of a pretype are different at every dimension: zero is a point in $\mathbb{I}$ but seg_x is not; on the other hand, both seg_x and zero (viewed degenerately) are lines in $\mathbb{I}$.

The above description, however, does not suffice as a definition, as it does not ensure the judgments are preserved by cubical operations. For example, if M is a line in A then its left face should be a point in $A\langle 0/x \rangle$—if $M \in A$ $[x]$ then $M\langle 0/x \rangle \in A\langle 0/x \rangle$ $[\cdot]$. This property is not automatic, because evaluation is also not closed under cubical operations: for instance, seg_x is canonical but seg_0 is not. Thus $A\langle 0/x \rangle$ may not be a pretype at all!

We solve this by demanding that any sequence of cubical operations applied to a pretype (resp., member) evaluate to a canonical pretype (resp., canonical member), and moreover, that the process of performing cubical operations then evaluating be functorial—for example, computing a face, evaluating, and computing a face of the result must agree with computing the iterated face at once.

To be precise, let $\psi : \Psi' \to \Psi$ denote any composite cubical operation taking Ψ-cubes to Ψ'-cubes. The judgment A $\mathsf{pretype}$ $[\Psi]$ states that for all $\psi : \Psi' \to \Psi$, A under ψ (written $A\psi$) evaluates to some A_0, and we know what are the canonical Ψ'-dimensional members of A_0 and when two such are equal. Furthermore, if ψ can be expressed as a composition of two cubical operations ψ_1 and ψ_2, then $A\psi_1$ evaluates to some A_1 which under ψ_2 evaluates to some A_2, and A_2 has the same canonical Ψ'-dimensional members as A_0. The judgment $M \in A$ $[\Psi]$ states that for all $\psi : \Psi' \to \Psi$, $M\psi$ and M under ψ_1 then ψ_2 (as before) evaluate to equal canonical Ψ'-dimensional members of A_0.

The equality judgments $A \doteq B$ $\mathsf{pretype}$ $[\Psi]$ and $M \doteq N \in A$ $[\Psi]$ are defined similarly, stating that A, B (resp., M, N) have *coherent cubical operations* as described above, and moreover, A_0 and B_0 have the same canonical members (resp., M_0 and N_0 are equal canonical members of A_0). The open judgments

$$a_1 : A_1, \ldots, a_n : A_n \gg A \doteq B \ \mathsf{pretype}\ [\Psi]$$
$$a_1 : A_1, \ldots, a_n : A_n \gg M \doteq N \in A\ [\Psi]$$

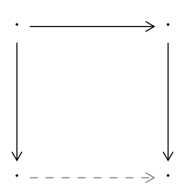
are defined by functionality, as before, with the caveat that functionality must hold at all dimension. That is, for any $\psi : \Psi' \to \Psi$ and any equal Ψ'-dimensional members of $A_1\psi, \ldots, A_n\psi$, the corresponding substitution instances of $A\psi, B\psi$

(resp., $M\psi, N\psi$) must be equal Ψ'-dimensional pretypes (resp., members). Notice that we consider open terms by instantiating each a_i with elements of A_i, but do not consider Ψ-dimensional cubes by instantiating each x_i with 0 or 1. Doing so equates all lines whose end points agree—in other words, it ensures uniqueness of identity proofs! It is critical that Ψ-cubes be treated as programs unto themselves, and first-class citizens of cubical type theory.

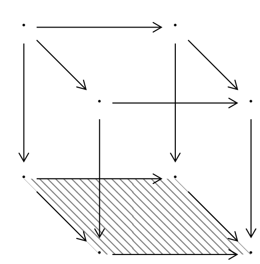
Once again, the role of types is to internalize mathematical statements about judgments. Most types retain their usual definition, parametrized over the dimension Ψ. For example, a canonical Ψ-dimensional member of the product pretype $A \times B$ $\textsf{pretype}$ $[\Psi]$ is a pair $\langle M, N\rangle$ of terms such that $M \in A$ $[\Psi]$ and $N \in B$ $[\Psi]$, and two canonical members $\langle M, N\rangle$ and $\langle M', N'\rangle$ are equal when $M \doteq M' \in A$ $[\Psi]$ and $N \doteq N' \in B$ $[\Psi]$.

The meaning of the identification pretype has changed dramatically. The identification pretype $\mathsf{Id}_{x.A}(M, N)$ $\textsf{pretype}$ $[\Psi]$ of A $\textsf{pretype}$ $[\Psi, x]$ has as canonical Ψ-dimensional members *abstracted lines* $\langle x\rangle P$ such that $P \in A$ $[\Psi, x]$ and moreover $P\langle 0/x\rangle \doteq M \in A\langle 0/x\rangle$ $[\Psi]$ and $P\langle 1/x\rangle \doteq N \in A\langle 1/x\rangle$ $[\Psi]$. Abstracting a dimension variable decreases the dimension of terms by one, so that an x-line P in A corresponds to a point $\langle x\rangle P$ in the identification type of A. This definition is not particularly complicated, nor should it be—in a sense the purpose of cubical judgments is to provide a judgmental notion of path for the identification type to straightforwardly internalize.

The cubical apparatus greatly simplifies the treatment of iterated path structure in higher-dimensional type theory, but we have not yet accounted for its groupoid structure—composition and inversion of paths, associativity and cancellation laws, and so forth. All of this structure can be imposed by means of a single family of operators establishing the *uniform Kan condition* [Bezem et al., 2014]. Loosely, the Kan condition (as originally defined by Kan [1955]) states that connected partial n-cubes extend to complete n-cubes: whenever three edges of a square exist, the fourth and interior do as well, and similarly for five faces of a cube:



(a)



(b)

Such partial n-cubes are, evocatively, called *open boxes*. The remaining side is called the *composite* of the open box, and the interior is the *filler*.

The uniform Kan condition invented by Bezem et al. [2014] adapts the Kan condition to type theory by means of two critical modifications. The first is to demand not only the classical existence of fillers, but an *operation* computing the filler of any open box. The second is to demand that this filling operation commutes with cubical operations. For instance, if we degenerate the open box (a), we obtain a partial box consisting of the top, left, and right faces of a cube; the composite of this degenerated box (its bottom face) must be the degeneracy of the composite of (a).

Miraculously, the groupoid structure of paths can be recovered from the Kan condition. Below we illustrate how to define path composition and inversion as composites of open boxes; recall that a point a can be regarded degenerately as a line from a to a. More complicated composites yield squares witnessing the unitality, cancellation, and associativity of the path operations defined this way.

$$\begin{tikzcd} a \arrow[r, "p"] \arrow[d, "a"'] & b \arrow[d, "q"] \\ a \arrow[r, dashed, "q \circ p"'] & c \end{tikzcd} \qquad \begin{tikzcd} a \arrow[r, "a"] \arrow[d, "p"'] & a \arrow[d, "a"] \\ b \arrow[r, dashed, "p^{-1}"'] & a \end{tikzcd}$$

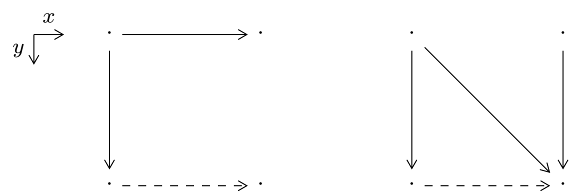
The Kan composition operations are rendered as programs in the cubical programming language, taking as arguments the faces of the open box and the type in which they lie. These operations compute according to their type argument; for instance, at type $A \times B$ one pairs the composite of the first projection of each face at type A, and the second projections at type B.

Types, then, are pretypes at which the Kan operations compute composites of open boxes. That is, the judgment A type $[\Psi]$ states that A $\mathsf{pretype}$ $[\Psi]$, and for every open box of members of A, the Kan operation at A applied to that box is a member of A that is the composite of that box.[2] The judgment $A \doteq B$ type $[\Psi]$ states that $A \doteq B$ $\mathsf{pretype}$ $[\Psi]$, and the Kan operations at A and B compute equal composites of open boxes in A. The fact that Kan operations are members of types ensures their uniformity (that they commute with cubical operations), because membership in a type is preserved by cubical operations, and cubical operations commute with term formers. Finally, the open typehood judgments are once again defined by functionality: all instances of an open type must be closed types, in a manner respecting equality.

Just as one might consider a variety of cubical operations, so one might consider a variety of open box shapes. It is critical that the Kan condition only apply to connected partial cubes, as opposed to, say, allowing one to fill a square given only its left and right edges. But must open boxes be symmetric? Must the open box include a face opposite from the composite? Cohen et al. [2016] allow computing the $\langle 1/y \rangle$ face of a filler, given its $\langle 0/y \rangle$ face and any number of faces with extent in the y direction. Angiuli et al. [2017] allow computing any y-face or y-diagonal of a filler, given any other y-face or y-diagonal and one or

[2] We are suppressing a technical side condition imposed in Angiuli et al. [2017], namely that the canonical members of A must also be members (i.e., have functorial cubical operations).

more pairs of opposing faces in other dimensions. Below are instances of exotic open boxes permitted by the former (c) and latter (d) type theories:



(c) (d)

The groupoid structure of higher inductive types is defined by freely adding Kan operations as canonical members. In the interval higher inductive type $\mathbb{I}$, zero and one are canonical Ψ-dimensional members for all Ψ (as they can be degenerated), seg_x is a canonical (Ψ, x)-dimensional member for all Ψ, and every Kan operation at $\mathbb{I}$ on members of $\mathbb{I}$ is also a canonical member at the appropriate dimension. These four clauses inductively define the canonical members of $\mathbb{I}$, and ensure by construction that $\mathbb{I}$ has Kan operations.

The univalence axiom of formal HoTT stipulates paths between types A and B for each homotopy equivalence E between A and B. Because a path between types in cubical type theory is itself a type, univalence gives rise to a type line $\mathsf{ua}_y(E)$ whose $\langle 0/y\rangle$ face is A and whose $\langle 1/y\rangle$ face is B. In formal HoTT, paths between types are equipped with an operation to *transport* elements of one end point to elements of the other; transport across an equivalence E applies that equivalence. In cubical type theory, transport is a special case of Kan composition in which the type varies in the direction of composition.[3] In the theory of Cohen et al. [2016], the $\mathsf{ua}_y(E)$ type is definable; Angiuli et al. [2017] define the type not_y, the instance of univalence corresponding to the equivalence between bool and itself swapping its points (as in Licata and Harper [2012]).

We close by outlining how the cubical meaning explanations subsume both the meaning explanations of computational type theory and the judgments of 2-dimensional formal type theory. If one restricts attention to cubical programs that are also ordinary programs—that is, programs with no dimension variables—then the cubical meaning explanations directly collapse to Martin-Löf's meaning explanations. This holds because the Kan operations do not affect this fragment, and the cubical coherence conditions described above are trivial for 0-cubes. Thus computational type theory can be seen simply as the lowest dimension of computational cubical type theory.

Similarly, the lowest two dimensions of computational cubical type theory correspond to the point and path judgments of 2-dimensional type theory. The judgment $a : A$ holds when $a \in A\ [\cdot]$, and $p : a \simeq_A b$ holds when $p \in A\ [x]$, $p\langle 0/x\rangle \doteq a \in A\ [\cdot]$, and $p\langle 1/x\rangle \doteq b \in A\ [\cdot]$. The composition and inverse path

[3] Angiuli et al. [2017] in fact decompose Kan composition into two operations: composition in constant types and transport.

operations can be implemented by Kan operations as outlined previously. The only wrinkle is that path operations behave strictly (up to definitional equality) in 2-dimensional type theory, and in cubical type theory one has squares witnessing the groupoid laws. Thus one must say two paths are equal, $p \equiv q : a \simeq_A b$, when there is a square identifying p and q.

5 Related and future work

Higher-dimensional type theory continues to be an active research area in multiple disciplines. Although cubical type theories resolve some shortcomings of formal HoTT, there are still reasons to study formal HoTT itself: to date, essentially all mechanized reasoning in higher-dimensional type theory uses theorem provers based on formal HoTT, and most research on categorical models of higher-dimensional type theories considers formal HoTT.

Although formal HoTT does not enjoy the canonicity property that every term $\cdot \vdash n : \mathbb{N}$ is definitionally equal to a numeral, Voevodsky [2011] has conjectured that one can always construct a path from n to a numeral. A positive resolution to this conjecture would recover some of the benefits of cubical type theory; for instance, it would resolve Brunerie's difficulty computing $\mathbb{Z}/2\mathbb{Z}$ [Brunerie, 2016]. Cubical type theory does not resolve this problem, because its identification type does not validate the so-called computation rule in formal HoTT for the identity eliminator on reflexive paths. As a result, proofs in formal HoTT have no direct interpretation in cubical type theory.

The concept of equality has long been a source of difficulty in formal type theory; prior to formal HoTT, without resorting to setoids [Barthe et al., 2003], one could neither quotient types by equivalence relations nor identify functions that agree in extension. These difficulties are resolved incidentally by higher inductive types and univalence, respectively, so these features potentially benefit all proofs in formal type theory. But one can also directly address these issues, by analyzing definable quotient types [Li, 2014], or developing a formal type theory admitting the principle of function extensionality [Altenkirch et al., 2007].

One can also add further axioms to formal HoTT, in order to capture additional structure of interest to homotopy theorists. Cohesive homotopy type theory allows types to have both path structure, as in formal HoTT, and topological structure, as in classical algebraic topology [Schreiber and Shulman, 2012]. Homotopy Type System extends formal HoTT with non-fibrant types that do not respect homotopy, including an exact equality type, and a fibrant replacement operation sending non-fibrant types to fibrant types [Voevodsky, 2013]. In the future, it may be possible to extend our higher-dimensional meaning explanations to accomodate these additional structures.

Cubical type theory, too, contains many open problems. At present, the authors' computational cubical type theory lacks a type universe and full univalence [Angiuli et al., 2017]. Cohen et al. [2016] have these features and have established a canonicity result [Huber, 2016], but other metatheoretic properties, such as decidability of type checking, are unknown. Very few higher inductive types have been defined in cubical type theory thus far, and general schemata for higher inductive types are not known, even in the context of formal HoTT. Cubical proof assistants remain in their infancy: computational cubical type theory is being implemented in the RedPRL proof assistant [Sterling et al., 2016], and Cohen et al. [2016] have implemented an experimental type checker.[4] Finally, the open-ended nature of meaning explanations permits all manner of constructions, including Brouwer's free choice sequences [Troelstra, 1973; Dummett, 1977] given by a purported creating subject rather than a computer program; the authors hope to pursue higher-dimensional analogues of such concepts in computational cubical type theory.

Acknowledgements

We are greatly indebted to Marc Bezem, Evan Cavallo, Kuen-Bang Hou (Favonia), Simon Huber, Dan Licata, Ed Morehouse, and Todd Wilson for their contributions and advice. A more technical account of the cubical meaning explanations described in this paper can be found in [Angiuli et al., 2017]. The authors gratefully acknowledge the support of the Air Force Office of Scientific Research through MURI grant FA9550-15-1-0053. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the AFOSR.

References

S. F. Allen. A Non-type-theoretic Definition of Martin-Löf's Types. In D. Gries, editor, *Proceedings of the 2nd IEEE Symposium on Logic in Computer Science*, pages 215–224. IEEE Computer Society Press, June 1987a.

S. F. Allen. *A Non-Type-Theoretic Semantics for Type-Theoretic Language*. PhD thesis, Cornell University, 1987b.

T. Altenkirch, C. McBride, and W. Swierstra. Observational equality, now. In *Programming Languages meets Program Verification Workshop*, 2007.

C. Angiuli, R. Harper, and T. Wilson. Computational higher-dimensional type theory. In *Proceedings of the 44th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '17, New York, NY, USA, 2017. ACM. To appear.

S. Awodey and M. A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1):45–55, Jan. 2009. ISSN 0305-0041. doi: 10.1017/S0305004108001783.

G. Barthe, V. Capretta, and O. Pons. Setoids in type theory. *Journal of Functional Programming*, 13(2):261–293, 03 2003.

[4] Their Haskell implementation can be found at `https://github.com/mortberg/cubicaltt`.

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

E. Bishop. *Foundations of constructive analysis*. McGraw-Hill Book Co., New York, 1967.

G. Brunerie. The James construction and $\pi_4(S^3)$, 2013. URL `https://video.ias.edu/univalent/1213/0327-GuillaumeBrunerie`. Video of a talk at the Institute for Advanced Study.

G. Brunerie. *On the homotopy groups of spheres in homotopy type theory*. PhD thesis, Université de Nice Sophia Antipolis, 2016. URL `http://arxiv.org/abs/1606.05916`.

C. Cohen, T. Coquand, S. Huber, and A. Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. In *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. To appear.

R. L. Constable and S. F. Smith. Computational foundations of basic recursive function theory. In *Proceedings of the 3^{rd} IEEE Symposium on Logic in Computer Science*, pages 360–371, Edinburgh, UK, 1988. IEEE Computer Society Press. (Cornell TR 88-904).

R. L. Constable and S. F. Smith. Computational foundations of basic recursive function theory. *Theoretical Comput. Sci.*, 121(1&2):89–112, Dec. 1993.

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

L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer. The Lean theorem prover (system description). In *CADE-25: 25th International Conference on Automated Deduction*. Springer International Publishing, 2015. ISBN 978-3-319-21401-6. doi: 10.1007/978-3-319-21401-6_26. URL `http://dx.doi.org/10.1007/978-3-319-21401-6_26`.

M. Dummett. *Elements of Intuitionism*. Oxford University Press, 1977.

N. Gambino and R. Garner. The identity type weak factorisation system. *Theoretical Computer Science*, 409(1):94 – 109, 2008. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/j.tcs.2008.08.030. URL `http://www.sciencedirect.com/science/article/pii/S0304397508006063`.

R. Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, second edition, 2016.

M. Hofmann and T. Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory*. Oxford University Press, 1998.

W. A. Howard. The formulae-as-types notion of construction. In J. R. Seldin, Jonathan P.; Hindley, editor, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.

D. J. Howe. Equality in lazy computation systems. In *Proceedings of the Fourth Annual IEEE Symposium on Logic in Computer Science (LICS 1989)*, pages 198–203. IEEE Computer Society Press, June 1989.

D. J. Howe and S. D. Stoller. An operational approach to combining classical set theory and functional programming languages. In a. J. C. M. M. Hahiya, editor, *Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 36–55, New York, April 1994. Springer, Berlin.

S. Huber. *Cubical Interpretations of Type Theory*. PhD thesis, University of Gothenburg, Expected November 2016.

D. M. Kan. Abstract homotopy. I. *Proceedings of the National Academy of Sciences of the United States of America*, 41(12):1092–1096, 1955. ISSN 00278424. URL `http://www.jstor.org/stable/89108`.

C. Kapulkin and P. L. Lumsdaine. The simplicial model of univalent foundations (after Voevodsky). Preprint, June 2016. URL `https://arxiv.org/abs/1211.2851`.

S. C. Kleene. *Introduction to Metamathematics*. van Nostrand, 1952.

N. Li. *Quotient types in type theory*. PhD thesis, University of Nottingham, September 2014. URL `http://eprints.nottingham.ac.uk/28941/1/Nuo%20Li%27s_Thesis.pdf`.

D. R. Licata and G. Brunerie. A cubical type theory, November 2014. URL `http://dlicata.web.wesleyan.edu/pubs/lb14cubical/lb14cubes-oxford.pdf`. Talk at Oxford Homotopy Type Theory Workshop.

D. R. Licata and R. Harper. Canonicity for 2-dimensional type theory. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, pages 337–348, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1083-3. doi: 10.1145/2103656.2103697. URL `http://doi.acm.org/10.1145/2103656.2103697`.

P. L. Lumsdaine. *Weak ω-Categories from Intensional Type Theory*, pages 172–187. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-02273-9. doi: 10.1007/978-3-642-02273-9_14. URL `http://dx.doi.org/10.1007/978-3-642-02273-9_14`.

P. Martin-Löf. An intuitionistic theory of types: predicative part. In H. Rose and J. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975.

P. Martin-Löf. Constructive mathematics and computer programming. In L. J. Cohen, J. o, H. Pfeiffer, and K.-P. Podewski, editors, *Logic, Methodology and Philosophy of Science VI, Proceedings of the Sixth International Congress of Logic, Methodology and Philosophy of Science, Hannover 1979*, volume 104 of *Studies in Logic and the Foundations of Mathematics*, pages 153–175. North-Holland, 1982. doi: 10.1016/S0049-237X(09)70189-2. URL `http://dx.doi.org/10.1016/S0049-237X(09)70189-2`.

P. Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, 1984. ISBN 88-7088-105-9. Notes by Giovanni Sambin of a series of lectures given in Padua, June 1980.

B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory*. Oxford University Press, 1990.

U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology, 2007.

V. Rahli and M. Bickford. Coq as a Metatheory for Nuprl with Bar Induction. Presented at *Continuity, Computability, Constructivity – From Logic to Algorithms (CCC 2015)*, 2015.

U. Schreiber and M. Shulman. Quantum gauge field theory in cohesive homotopy type theory. Quantum Physics and Logic 2012, October 2012. URL `https://arxiv.org/abs/1408.0054v1`.

M. Shulman. Homotopy type theory, VI, 2011. URL `https://golem.ph.utexas.edu/category/2011/04/homotopy_type_theory_vi.html`. Blog post on the n-category café.

J. Sterling, D. Gratzer, V. Rahli, D. Morrison, E. Akentyev, and A. Tosun. RedPRL – the People's Refinement Logic. `http://www.redprl.org/`, 2016.

T. Streicher. Identity types vs. weak omega-groupoids: some ideas, some problems. Talk given in Uppsala at the meeting on "Identity Types: Topological and Categorical Structure", 2006. URL `http://www.mathematik.tu-darmstadt.de/~streicher/TALKS/uppsala.pdf.gz`.

G. Sundholm. Constructions, proofs and the meaning of logical constants. *Journal of Philosophical Logic*, 12(2):151–172, 1983. ISSN 00223611, 15730433. URL `http://www.jstor.org/stable/30226268`.

The Coq Project. The Coq proof assistant, 2016. URL `http://www.coq.inria.fr`.

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

A. S. Troelstra. *Metamathematical Investigations of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer, 1973.

A. S. Troelstra and D. van Dalen. *Constructivism in mathematics. Vol. I*, volume 121 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam, 1988. ISBN 0-444-70266-0; 0-444-70506-6. An introduction.

B. van den Berg and R. Garner. Types are weak ω-groupoids. *Proceedings of the London Mathematical Society*, 102(2):370–394, 2011. doi: 10.1112/plms/pdq026. URL `http://plms.oxfordjournals.org/content/102/2/370.abstract`.

V. Voevodsky. A very short note on homotopy λ-calculus, 09 2006. URL `http://www.math.ias.edu/vladimir/files/2006_09_Hlambda.pdf`.

V. Voevodsky. The equivalence axiom and univalent models of type theory, 2010. URL `http://www.math.ias.edu/vladimir/files/CMU_talk.pdf`. Notes from a talk at Carnegie Mellon University.

V. Voevodsky. Univalent foundations of mathematics. In *Logic, Language, Information and Computation - 18th International Workshop, WoLLIC 2011, Philadelphia, PA, USA, May 18-20, 2011. Proceedings*, page 4, 2011. doi: 10.1007/978-3-642-20920-8_4. URL `https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/2011_WoLLIC.pdf`.

V. Voevodsky. A simple type system with two identity types. Lecture notes, Feb. 2013. URL `https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/HTS.pdf`.

V. Voevodsky, B. Ahrens, D. Grayson, et al. *UniMath*: Univalent Mathematics. Available at `https://github.com/UniMath`.