Thesis Proposal Computational Higher-Dimensional Type Theory

Carlo Angiuli

May 2017

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Robert Harper, Chair
Jeremy Avigad
Karl Crary
Daniel R. Licata, Wesleyan University
Todd Wilson, California State University, Fresno

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

May 10, 2017

Keywords: homotopy type theory, computational type theory, dependent type theory, logical relations, meaning explanations, cubical sets

Abstract

Intuitionistic type theory is a widely-used framework for constructive mathematics and computer programming. Martin-Löf's meaning explanations justify the rules of type theory by defining types as classifications of programs according to their behaviors. Recently, 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 such higher-dimensional type theories have proved useful for homotopy-theoretic reasoning, they lack a computational interpretation.

In this proposal, I describe a modification of the meaning explanations that accounts for higher-dimensional types as classifications of higher-dimensional programs. I aim to extend these cubical meaning explanations with additional features inspired by other higher-dimensional and computational type theories, and explore the programming applications of higher-dimensional programs.

Contents

1 Introduction

Martin-Löf's intuitionistic type theory is a comprehensive theory of constructions intended as a framework for constructive mathematics and computer programming [43, 44, 45]. How is it that such apparently disparate objectives could be achieved by the same framework? A guiding principle of type theory is the proofs-as-programs correspondence [30] that relates logical propositions to types, and proofs of propositions to programs of certain types. This tight connection between computer science and logic makes type theory especially well-suited to the mechanization of mathematics; indeed, many theorem provers, including Nuprl [57], Coq [56], Agda [47], and Lean [22], are built on variations of Martin-Löf's type theory.

The field of *homotopy type theory* (HoTT) has its origins in the observation that certain type theories admit homotopy-theoretic models in groupoids [27], Quillen model categories [11], weak factorization systems [24], weak ω-groupoids [41, 59], and simplicial sets [55, 60]. In such models, types are interpreted as spaces, their elements as points, and (n-fold) proofs of identity as (n-dimensional) paths.

These *higher-dimensional* models allow one to use type theory as a synthetic framework for homotopy-theoretic reasoning. Accordingly, researchers have augmented Martin-Löf's type theory with axioms valid in these models: *higher inductive types* (HITs), inductive types generated by not only points but also paths [50]; and Voevodsky's *univalence axiom*, stating that identity of types is homotopy equivalence (a higher-dimensional analogue of isomorphism) [35, 61].

As a practical matter, the resulting theory—which I term "book HoTT," due to its popularization by the HoTT Book [58]—has already achieved success proving results about spaces [58], including the Seifert-van Kampen [28] and Blakers-Massey [29] theorems, among others. Philosophically, however, the haphazard introduction of additional axioms to type theory is troubling; these axioms are only justified through appeals to mathematical models defined with classical logic. Computationally, the situation is worse yet: the proofs-as-programs correspondence is not known to extend to book HoTT, eliminating many of type theory's unique strengths.

My contributions, the *cubical meaning explanations*, extend the meaning explanations of type theory in order to properly account for higher-dimensional structure. This extension is founded on two key insights: that the judgmental apparatus of type theory must itself be generalized to higher dimension [39], and that *abstract cubes* afford a convenient syntactic representation of higher-dimensional structure [13]. The cubical meaning explanations establish a precise connection between higher-dimensional type theory and higher-dimensional programming, namely, that:

> Higher-dimensional types classify higher-dimensional programs extensionally according to their behaviors.

Philosophically, this connection explains the higher-dimensional content of the logical connectives. Semantically, it forms a syntactic model of higher-dimensional type theory equipped with a well-defined notion of extensional equality. In this thesis proposal, I argue that my current work can be extended to a variety of features inspired by other higher-dimensional type theories, computational type theories, and programming applications.

1.1 Acknowledgements

In addition to my thesis committee members, I would like to especially thank Steve Awodey, Marc Bezem, Guillaume Brunerie, Evan Cavallo, Daniel Gratzer, Kuen-Bang Hou (Favonia), Simon Huber, Ed Morehouse, and Jon Sterling for their helpful advice.

Portions of this document have previously appeared in Angiuli et al. [7], Angiuli et al. [9], Angiuli and Harper [6], Angiuli et al. [10], and Angiuli and Harper [5]. These papers were sponsored by the National Science Foundation under grant number CCF-1116703, and by 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 any sponsoring institution, the U.S. government or any other entity.

2 Background and Prior Work

In this section, I summarize the basic features of homotopy type theory and various limitations of its standard formulation, most notably its incompatibility with the meaning explanations of type theory and its lack of a proofs-as-programs interpretation. I then summarize my work on computational higher-dimensional type theory, and explain how it addresses these limitations.

2.1 Higher-Dimensional Type Theory

Many versions of type theory are inductively defined by a collection of inference rules specifying an assortment of types ($\Gamma \vdash A\ \mathsf{type}$) and their inhabitants ($\Gamma \vdash a : A$), and when two types or inhabitants are to be considered definitionally equal ($\Gamma \vdash A \equiv B\ \mathsf{type}$, $\Gamma \vdash a \equiv b : A$). Certain such type theories are equipped with an *identity type* $a =_A b$ of proofs that $a : A$ and $b : A$ are equal, defined in part by the rules [43, 58]:

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

The introduction rule (left) establishes that every $a : A$ is identical to itself by refl_a, and the elimination rule (right) 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 p, q of $a =_A b$ necessarily identical? Traditional models of type theory validate this property, known as *uniqueness of identity proofs*, but Hofmann and Streicher [27] showed that it does not follow from the typical axioms of 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, $p : a =_A b$ is a morphism (or path) between a and b in A, and $f : A \to B$ is a functor from A to B. Uniqueness of identity proofs fails here because parallel morphisms need not be equal.

Subsequent work on homotopy-theoretic models of type theory further generalized the Hofmann–Streicher groupoid model to infinite-dimensional structures, in which $\alpha : p =_{(a =_A b)} q$ is a 2-morphism between the parallel 1-morphisms p and q, *et cetera*. Such infinite-dimensional structures, most notably weak ω-groupoids and simplicial sets, are primarily used by mathematicians as algebraic presentations of spaces. As a result, type theory can be seen as an axiomatic, or synthetic, framework for reasoning about spaces.

However, because type theory was not originally intended for that purpose, none of its types have interesting higher-dimensional structures. In particular, n-morphisms cannot be proven to be nontrivial, due to the consistency of uniqueness of identity proofs. One must instead build *higher-dimensional type theories*,[1] type theories extended with novel principles inspired by higher-dimensional models. The most widely-used higher-dimensional type theory, book HoTT, adds higher inductive types and the univalence axiom [58]; I briefly summarize these features below.

Higher inductive types are a form of inductive type whose generators populate not only the type itself but also its (iterated) identity types. A simple example is the *interval type*, defined by the rules:

$$\frac{}{\Gamma \vdash \mathbb{I}\ \mathsf{type}} \qquad \frac{}{\Gamma \vdash \mathsf{0}_{\mathbb{I}} : \mathbb{I}} \qquad \frac{}{\Gamma \vdash \mathsf{1}_{\mathbb{I}} : \mathbb{I}} \qquad \frac{}{\Gamma \vdash \mathsf{seg} : \mathsf{0}_{\mathbb{I}} =_{\mathbb{I}} \mathsf{1}_{\mathbb{I}}}$$

$$\frac{\begin{array}{c}\Gamma, x : \mathbb{I} \vdash C\ \mathsf{type} \\ \Gamma \vdash a : C[\mathsf{0}_{\mathbb{I}}/x] \\ \Gamma \vdash b : C[\mathsf{1}_{\mathbb{I}}/x] \\ \Gamma \vdash s : a =^{x.C}_{\mathsf{seg}} b \\ \Gamma \vdash y : \mathbb{I}\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{I}}(x.C, a, b, s, y) : C[y/x]} \qquad \frac{\begin{array}{c}\Gamma, x : \mathbb{I} \vdash C\ \mathsf{type} \\ \Gamma \vdash a : C[\mathsf{0}_{\mathbb{I}}/x] \\ \Gamma \vdash b : C[\mathsf{1}_{\mathbb{I}}/x] \\ \Gamma \vdash s : a =^{x.C}_{\mathsf{seg}} b\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{I}}(x.C, a, b, s, \mathsf{0}_{\mathbb{I}}) \equiv a : C[\mathsf{0}_{\mathbb{I}}/x]}$$

$$\frac{\begin{array}{c}\Gamma, x : \mathbb{I} \vdash C\ \mathsf{type} \\ \Gamma \vdash a : C[\mathsf{0}_{\mathbb{I}}/x] \\ \Gamma \vdash b : C[\mathsf{1}_{\mathbb{I}}/x] \\ \Gamma \vdash s : a =^{x.C}_{\mathsf{seg}} b\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{I}}(x.C, a, b, s, \mathsf{1}_{\mathbb{I}}) \equiv b : C[\mathsf{1}_{\mathbb{I}}/x]} \qquad \frac{\begin{array}{c}\Gamma, x : \mathbb{I} \vdash C\ \mathsf{type} \\ \Gamma \vdash a : C[\mathsf{0}_{\mathbb{I}}/x] \\ \Gamma \vdash b : C[\mathsf{1}_{\mathbb{I}}/x] \\ \Gamma \vdash s : a =^{x.C}_{\mathsf{seg}} b\end{array}}{\Gamma \vdash \mathsf{seg}_{\beta} : \mathsf{apd}_{\lambda y.\mathsf{ind}_{\mathbb{I}}(x.C,a,b,s,y)}(\mathsf{seg}) = s}$$

The introduction rules stipulate that $\mathbb{I}$ has two objects, $\mathsf{0}_{\mathbb{I}}$ and $\mathsf{1}_{\mathbb{I}}$, and a path seg between them. The elimination rule constructs a map from $\mathbb{I}$ to a type family $C(x)$, given objects $a : C(\mathsf{0}_{\mathbb{I}})$ and $b : C(\mathsf{1}_{\mathbb{I}})$ and a path s between them. Because a and b have different types, s is not an ordinary path but rather a *dependent path* $a =^{x.C}_{\mathsf{seg}} b$ in the family $C(x)$ over seg [58, Equation 6.2.2]. Maps constructed by the elimination rule definitionally send $\mathsf{0}_{\mathbb{I}}$ and $\mathsf{1}_{\mathbb{I}}$ to a and b respectively. The elimination rule does not apply to seg directly, as it is not an element of $\mathbb{I}$ but rather of $\mathsf{0}_{\mathbb{I}} =_{\mathbb{I}} \mathsf{1}_{\mathbb{I}}$; however, when lifted to an operation on paths by $\mathsf{apd}(-)$, such maps applied to seg are related by a path to s [58, Lemma 2.3.4].

Voevodsky's univalence axiom, the other addition of book HoTT, states that a particular map from paths between types to homotopy equivalences is itself a homotopy equivalence:

$$\frac{\Gamma \vdash A : \mathcal{U}_i \quad \Gamma \vdash B : \mathcal{U}_i}{\Gamma \vdash \mathsf{univalence}(A, B) : \mathsf{isequiv}(\mathsf{idtoeqv}_{A,B})}$$

[1]The term "homotopy type theory" (or HoTT) refers variously to the study of connections between homotopy theory and type theory; to any higher-dimensional type theory; or to book HoTT specifically. I will use it only in the first sense, to avoid confusion.

Equivalence ($\mathsf{isequiv}(-)$) can be defined in ordinary type theory using dependent function, dependent pair, and identity types [58, Definition 4.2.1]; informally, an equivalence between A and B is a function from $A \to B$ that is invertible modulo paths. The function $\mathsf{idtoeqv}_{A,B}$, also definable in ordinary type theory, sends paths $A = B$ to equivalences between A and B [58, Lemma 2.10.1]. Because all constructions in type theory respect identity, the force of univalence is to stipulate that *all constructions respect equivalence* as well.

The theorem provers Agda, Coq, and Lean have been extended with higher inductive types and the univalence axiom, allowing researchers to mechanize proofs in book HoTT. However, the rules of book HoTT have a markedly different character from the ordinary rules of type theory. The rules defining $\mathbb{I}$ populate $\mathbb{I}$, the identity type of $\mathbb{I}$, and arbitrary dependent path types (via seg_β), without altering the elimination principle of the identity type to account for these additional identity elements. As a result, book HoTT lacks the *canonicity property*—typical of type theories—that any closed term of type $\mathbb{N}$ is definitionally equal to a numeral: if $\cdot \vdash n : \mathbb{N}$ then $\cdot \vdash n \equiv \mathsf{s}^m(\mathsf{z}) : \mathbb{N}$.

2.2 Canonicity and Definitional Equality

To understand why canonicity is desirable, we must first understand the role of the definitional equality judgment in type theories inductively defined by inference rules. Such type theories need not specify a notion of definitional equality at all, although it is required for the resulting logic to be practical [20].

Logically, the judgment $\Gamma \vdash a : A$ expresses that a is a *proof term* for the proposition A, under assumptions Γ. Two proofs are typically considered definitionally equal, $\Gamma \vdash a \equiv b : A$, when $\Gamma \vdash a : A$, $\Gamma \vdash b : A$, and a, b are α, β-equivalent as λ-terms [20].[2] Definitional equality of types, $\Gamma \vdash A \equiv B\ \mathsf{type}$, is generated by definitional equality of terms, in the sense that the following rule is admissible:

$$\frac{\Gamma, x : A \vdash B\ \mathsf{type} \quad \Gamma \vdash a \equiv a' : A}{\Gamma \vdash B[a/x] \equiv B[a'/x]\ \mathsf{type}}$$

The formal purpose of definitional equality is to give terms more types:

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash A \equiv B\ \mathsf{type}}{\Gamma \vdash a : B}$$

This rule stipulates, for instance, that if P is a property of natural numbers, a proof of $P(1+1)$ is precisely a proof of $P(2)$ as well (because $\cdot \vdash 1+1 \equiv 2 : \mathbb{N}$). Notice that application of this rule is not marked in the proof term a! In contrast, while the path $\Gamma \vdash \mathsf{seg} : 0_{\mathbb{I}} = 1_{\mathbb{I}}$ allows us to compare $\Gamma \vdash a : C(0_{\mathbb{I}})$ and $\Gamma \vdash b : C(1_{\mathbb{I}})$ despite their types differing, we nevertheless do not consider a an object of $C(1_{\mathbb{I}})$.

[2]Some presentations also include η-equivalence at function and product types, although these are regarded as optional in book HoTT [58, Remark 1.5.1].

The restriction of definitional equality to α, β-equivalence—rather than, say, extensional equality of λ-terms, or equality in some model—is born of the desire to make proof verification *decidable*: that is, one can decide whether $\Gamma \vdash a : A$ is derivable, and if so, construct a derivation. In turn, this licenses the reading that a is a proof term for A, as a itself contains enough information to construct a derivation of A.

The decision procedure is relatively straightforward. In most cases, the proof rule used to derive $\Gamma \vdash a : A$ is determined by the outermost constructor of a: for instance, implication introduction is marked by λ. If a definitional equality $\Gamma \vdash A \equiv B\ \mathsf{type}$ is needed to derive the judgment, then one must decide it by checking that $\Gamma \vdash A\ \mathsf{type}$, $\Gamma \vdash B\ \mathsf{type}$, and that A and B normalize to the same term.

Because book HoTT lacks canonicity, proofs using higher inductive types and univalence require additional bookkeeping to mark uses of identity proofs. Although it may sound minor, this defect has major consequences for users of book HoTT. In 2013, Brunerie proved that a particular topological invariant is given by the group $\mathbb{Z}/n\mathbb{Z}$ where $\cdot \vdash n : \mathbb{N}$ [14]. If canonicity held, one could simply replace the n in the theorem with a definitionally equal numeral. Instead, Brunerie was not able to prove that this invariant is $\mathbb{Z}/2\mathbb{Z}$ until 2016, after developing a second, more elaborate proof [15].

In theory, one might be able to regain canonicity by adding definitional equalities to book HoTT. In practice, such a formulation of definitional equality would likely depart from $\alpha, \beta(, \eta)$-equivalence on proof terms, and it is not clear which equations are validated by all desired higher-dimensional models of book HoTT.[3] We certainly cannot turn all paths into definitional equalities—by univalence, there is a path between the types $A \times B$ and $B \times A$; if these were definitionally equal, then any term of type $A \times B$ would also have type $B \times A$. For these and other reasons, no major attempts at formulating a higher-dimensional type theory with canonicity have proceeded along these lines [2, 10, 16, 38, 39].

However, the failure of canonicity in book HoTT is also symptomatic of a deeper philosophical concern. The canonicity property, the calculational nature of definitional equality, and the standard proofs-as-programs interpretation of type theory all have their roots in Martin-Löf's original *meaning explanations* of the judgments of type theory [44], which I summarize in Section 2.3.

These meaning explanations, and the standard proofs-as-programs interpretation, do not extend in any obvious way to book HoTT. Adding proof rules to type theory without meaning-theoretic justification disrupts the philosophical justification of *all* rules of type theory. Furthermore, the lack of a proof-as-programs interpretation makes it difficult for computer scientists to seriously discuss programming applications of higher-dimensional type theory [9, 12, 23], as these programs do not run!

[3]Various researchers have conjectured that all such models are elementary $(\infty, 1)$-toposes, but only recently has a concrete definition been proposed [51].

2.3 Martin-Löf's Meaning Explanations

I will briefly recall Martin-Löf's meaning explanations of type theory [1, 19, 44], before describing how to extend them to higher dimension. 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):

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

I have switched notation to $\in$ and $\doteq$ (and later, $\gg$), to emphasize that these judgments will have a *different intended meaning* than the judgments I have previously discussed, notated with :, $\equiv$, and $\vdash$.

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.[4] In fact, one can even consider classical set-theoretic functions as programs [32].

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* [46] and *abstract binding trees* [26]. Expressions M are given computational meaning by specifying which are *canonical* (M val), or not subject to further computation; and by defining an *operational semantics* ($M \longmapsto M'$) 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 [18].

We first specify the meanings of the basic judgments A 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.

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

[4]The NuPRL computational type theory adopts a mild constraint on possible programs introduced by Howe [31], in the interest of increasing the utility of the theory. To date, no proposed extension of the theory has run afoul of this constraint.

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 $\mathbb{N} \to \mathbb{N}$, but are of course distinct functions $\mathbb{Z} \to \mathbb{Z}$. In fact, as a matter of technical convenience, one can define only the binary forms of judgment, and recover $A\ \mathsf{type}$ 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*:

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

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, $a_1 : A_1 \gg A \doteq B\ \mathsf{type}$ states that for any $M \doteq N \in A_1$, we know $A[M/a_1] \doteq B[N/a_1]\ \mathsf{type}$. 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. (In this sense, 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 $a : A \gg M \doteq M' \in B$. Such definitions proceed in a predicative fashion, in which successive definitions build on prior ones.

The meaning explanation of the identity type $M =_A N$ is that refl_M is a canonical member whenever $M \doteq N \in A$, that is, it internalizes the equality judgment. This explanation is incompatible with the intended meaning of the identity type in book HoTT, namely, that it represents paths in A.

The meaning explanations *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 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 the meaning explanations to serve as a window on the full truth. The type theory on which book HoTT is based is one such fragment—an inductively-defined *proof theory* directly suitable for implementation in a proof assistant—in which all proof rules are either decidable or marked in the proof term (as discussed in Section 2.2). Henceforth, I will refer to inductively-defined type theories as *formal type theories*, and to meaning explanations as *computational type theories* [17] due to their direct basis in computation.

2.4 Computational Higher-Dimensional Type Theory

Computational higher-dimensional type theory is described in Angiuli and Harper [6] and Angiuli et al. [10]. I briefly summarize the core points below.

Following Martin-Löf's methodology that types internalize judgments, a proper accounting of higher-dimensional type theory requires judgments for paths, paths between paths, *et cetera.* Following Bezem et al. [13], we present this structure by means of a *cubical* generalization of both the judgments of computational type theory and the programming language on which it is based.

The cubical programming language has two sorts—ordinary terms ($A, B, M, N, \dots$ with variables $a, b, \dots$) and *dimension terms* (r, r'), which are either 0 or 1 (ε), or a *dimension name* ($x, y, z, \dots$). Dimension names are nominal constants [48, 49] representing formal elements of an interval with endpoints 0 and 1. Dimension terms appear in certain term formers; for example, loop_r is a program for any dimension term r. We define a *dimension substitution* operation $M\langle r/x\rangle$ which replaces free occurrences of x in M with r, alongside ordinary term substitution written $M[N/a]$.

We write $\mathsf{FD}(M)$ for the set of dimension names free in M, and say that M is a Ψ*-cube* if $\mathsf{FD}(M) \subseteq \Psi$. An x-cube M represents an abstract line in the x direction, whose left endpoint, or *face*, is a $\emptyset$-cube (or point) $M\langle 0/x\rangle$, and whose right face is $M\langle 1/x\rangle$. An (x, y)-cube N is an abstract square with four lines as its boundary, and four points as the boundary of those lines:

$$\begin{tikzcd}
{} \arrow[r, "x"] \arrow[d, "y"'] & {} \\
{} &
\end{tikzcd}
\qquad
\begin{tikzcd}
N\langle 0/x\rangle\langle 0/y\rangle \arrow[r, "N\langle 0/y\rangle"] \arrow[d, "N\langle 0/x\rangle"'] \arrow[dr, phantom, "N"] & N\langle 1/x\rangle\langle 0/y\rangle \arrow[d, "N\langle 1/x\rangle"] \\
N\langle 0/x\rangle\langle 1/y\rangle \arrow[r, "N\langle 1/y\rangle"'] & N\langle 1/x\rangle\langle 1/y\rangle
\end{tikzcd}$$

The fact that dimension substitutions commute validates the geometrical fact that the $\langle 0/x\rangle$ face of $N\langle 0/y\rangle$ must agree with the $\langle 0/y\rangle$ face of $N\langle 0/x\rangle$ in the upper left.

A Ψ-cube M can be regarded trivially as a *degenerate* (Ψ, x)-cube whose x-faces are both M. Finally, we can substitute one dimension name for another, which takes the *diagonal* of a square. $N\langle x/y\rangle$ is an x-line (the upper-left-to-lower-right diagonal in the diagram above) whose left face is $N\langle 0/x\rangle\langle 0/y\rangle$ and whose right face is $N\langle 1/x\rangle\langle 1/y\rangle$. We call all combinations of faces, diagonals, and degeneracies of a cube its *aspects*.

Aspects are obtained by means of total dimension substitutions, written $\psi : \Psi' \to \Psi$, which take a Ψ-cube M to a Ψ'-cube $M\psi$.

Our cubical programming language has the following terms:

$$\begin{aligned} M := {} & (a{:}A) \to B \mid (a{:}A) \times B \mid \mathsf{Id}_{x.A}(M,N) \mid \mathsf{bool} \mid \mathsf{not}_r \mid \mathbb{S}^1 \mid \mathsf{sbool} \mid \mathsf{ia}_r(A,B,M,N) \\ & \mid \lambda a.M \mid \mathsf{app}(M,N) \mid \langle M,N\rangle \mid \mathsf{fst}(M) \mid \mathsf{snd}(M) \mid \langle x\rangle M \mid M@r \\ & \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}_{a.A}(M;N_1,N_2) \mid \mathsf{notel}_r(M) \mid \mathsf{iain}_r(M,N) \mid \mathsf{iaout}_r(M,N) \\ & \mid \mathsf{base} \mid \mathsf{loop}_r \mid \mathbb{S}^1\text{-}\mathsf{elim}_{a.A}(M;N_1,x.N_2) \mid \mathsf{coe}^{r\rightsquigarrow r'}_{x.A}(M) \mid \mathsf{hcom}^{\overrightarrow{r_i}}_{A}(r \rightsquigarrow r', M; \overrightarrow{y.N^\varepsilon_i}) \end{aligned}$$

$\langle x\rangle M$ abstracts the dimension x in M, and $M@r$ applies such an abstraction to a dimension term. $\mathbb{S}^1$ is the higher inductive type corresponding to *the circle*, which has a base point base and an x-line loop_x. Just as if takes a motive type $a.A$, a boolean M, and two cases N_1, N_2 corresponding to true and false, the eliminator for the circle takes two cases N_1 and $x.N_2$ corresponding to its generating base point and line. not_r is an instance of univalence corresponding to the negation isomorphism between bool and itself, and $\mathsf{ia}_r(A,B,M,N)$ is an instance of univalence for strict isomorphisms. Finally, coe and hcom implement the *Kan operations*, which I describe later.

As before, we define an operational semantics by specifying the canonical terms (M val) and a deterministic weak head reduction ($M \longmapsto M'$), and we say $M \Downarrow V$ when $M \longmapsto^* V$ and V val. (The full operational semantics can be found in Angiuli and Harper [6, Section 2.2].) These judgments apply to closed terms of *any dimension*, that is, terms containing free dimension names but not free term variables. Some steps are responsible for calculating the faces of terms:

$$\frac{}{\mathsf{base}\ \mathsf{val}} \qquad \frac{}{\mathsf{loop}_x\ \mathsf{val}} \qquad \frac{}{\mathsf{loop}_0 \longmapsto \mathsf{base}} \qquad \frac{}{\mathsf{loop}_1 \longmapsto \mathsf{base}}$$

Here, base and loop_x are canonical terms (and canonical elements of $\mathbb{S}^1$); the x-faces of loop_x are both base, because both $(\mathsf{loop}_x)\langle 0/x\rangle$ and $(\mathsf{loop}_x)\langle 1/x\rangle$ calculate to base. At higher types, we include familiar β-reductions:

$$\frac{M \longmapsto M'}{\mathsf{app}(M,N) \longmapsto \mathsf{app}(M',N)} \qquad \frac{}{\mathsf{app}(\lambda a.M, N) \longmapsto M[N/a]} \qquad \frac{}{\lambda a.M\ \mathsf{val}}$$

$$\frac{}{\mathsf{hcom}^{\overrightarrow{r_i}}_{(a:A)\to B}(r \rightsquigarrow r', M; \overrightarrow{y.N^\varepsilon_i}) \longmapsto \lambda a.\mathsf{hcom}^{\overrightarrow{r_i}}_{B}(r \rightsquigarrow r', \mathsf{app}(M,a); \overrightarrow{y.\mathsf{app}(N^\varepsilon_i, a)})}$$

$$\frac{}{\mathsf{coe}^{r\rightsquigarrow r'}_{x.(a:A)\to B}(M) \longmapsto \lambda a.\mathsf{coe}^{r\rightsquigarrow r'}_{x.B[\mathsf{coe}^{r'\rightsquigarrow x}_{x.A}(a)/a]}(\mathsf{app}(M, \mathsf{coe}^{r'\rightsquigarrow r}_{x.A}(a)))}$$

as well as steps implementing the Kan operations.

The four basic judgments of computational cubical type theory express typehood and membership at dimension Ψ:

$$\begin{array}{cc} A \ \mathsf{pretype}\ [\Psi] & A \doteq B \ \mathsf{pretype}\ [\Psi] \\ M \in A\ [\Psi] & M \doteq N \in A\ [\Psi] \end{array}$$

I define these judgments relative to a pair of indexed partial equivalence relations (PERs) called a *cubical type system*, in a generalization of the NuPRL semantics [1, 25]. A cubical type system is:

- For every Ψ, a symmetric and transitive relation $\approx^{\Psi}$ over values A_0 with $\mathsf{FD}(A_0) \subseteq \Psi$, specifying the (equal) canonical Ψ-dimensional pretypes, and
- For every $A_0 \approx^{\Psi} B_0$, equal symmetric and transitive relations $\approx^{\Psi}_{A_0}$ and $\approx^{\Psi}_{B_0}$ over values M_0 with $\mathsf{FD}(M_0) \subseteq \Psi$, specifying the (equal) canonical Ψ-dimensional elements of A_0 and B_0.

The meanings of the cubical judgments are complicated by the requirement that they be closed under dimension substitution. We say that $A \doteq B$ $\mathsf{pretype}$ $[\Psi]$, presupposing that $\mathsf{FD}(A, B) \subseteq \Psi$, when for any $\psi_1 : \Psi_1 \to \Psi$ and $\psi_2 : \Psi_2 \to \Psi_1$,

- $A\psi_1 \Downarrow A_1$, $A_1\psi_2 \Downarrow A_2$, $A\psi_1\psi_2 \Downarrow A_{12}$,
- $B\psi_1 \Downarrow B_1$, $B_1\psi_2 \Downarrow B_2$, $B\psi_1\psi_2 \Downarrow B_{12}$, and
- $A_2 \approx^{\Psi_2} A_{12} \approx^{\Psi_2} B_2 \approx^{\Psi_2} B_{12}$.

and that $M \doteq N \in A$ $[\Psi]$, presupposing $A \doteq A$ $\mathsf{pretype}$ $[\Psi]$ and $\mathsf{FD}(M, N) \subseteq \Psi$, when for any $\psi_1 : \Psi_1 \to \Psi$ and $\psi_2 : \Psi_2 \to \Psi_1$,

- $M\psi_1 \Downarrow M_1$, $M_1\psi_2 \Downarrow M_2$, $M\psi_1\psi_2 \Downarrow M_{12}$,
- $N\psi_1 \Downarrow N_1$, $N_1\psi_2 \Downarrow N_2$, $N\psi_1\psi_2 \Downarrow N_{12}$, and
- $M_2 \approx^{\Psi_2}_{A_{12}} M_{12} \approx^{\Psi_2}_{A_{12}} N_2 \approx^{\Psi_2}_{A_{12}} N_{12}$, where $A\psi_1\psi_2 \Downarrow A_{12}$.

We write A $\mathsf{pretype}$ $[\Psi]$ when $A \doteq A$ $\mathsf{pretype}$ $[\Psi]$, and $M \in A$ $[\Psi]$ when $M \doteq M \in A$ $[\Psi]$.

These conditions guarantee that when M is a Ψ-cube of A, not only does M evaluate to a canonical Ψ-cube of A, but the $\psi : \Psi' \to \Psi$ aspect of M evaluates to a canonical Ψ'-cube of $A\psi$, and its $\psi_1\psi_2$ aspect and the ψ_2 aspect of its ψ_1 aspect's value evaluate to equal canonical Ψ_2-cubes of A. We say this ensures M has *coherent aspects*. Note that this coherence is not guaranteed by the operational semantics; it is the role of the type system to carve out the sensible programs.

We say A $\mathsf{pretype}$ $[\Psi]$ is *cubical* if for any $\psi : \Psi' \to \Psi$ and $M \approx^{\Psi'}_{A_0} N$ (where $A\psi \Downarrow A_0$), $M \doteq N \in A\psi$ $[\Psi']$. In a cubical pretype, the values of any element's aspects again have coherent aspects, and hence arbitrary interleavings of dimension substitutions and evaluation are coherent.

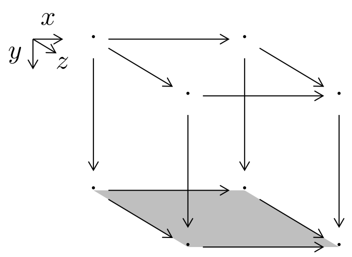
Finally, when restricted to terms M with no free dimensions, the cubical meaning explanations collapse to the ordinary ones: $A \doteq B$ $\mathsf{pretype}$ $[\Psi]$ when $A \Downarrow A_0$, $B \Downarrow B_0$, and $A_0 \approx^{\Psi'} B_0$ for all Ψ'.

The cubical apparatus does not yet account for the groupoid structure of paths—composition and inversion of paths, associativity and cancellation laws, *et cetera*—

present in book HoTT. This structure is imposed by two Kan operations. The first, called *coercion* (coe), takes an x-line A between types and an element M of $A\langle r/x\rangle$ to an element of $A\langle r'/x\rangle$. The second, called *homogeneous Kan composition* (hcom), approximately requires that open boxes in any type have lids.

The simplest composition scenario states that a U-shaped configuration of lines always forms the boundary to a square. If M is an x-line in A, and $y.N^\varepsilon$ are y-lines in A, and their endpoints agree as depicted below (left), then $\mathsf{hcom}^x_A(0 \rightsquigarrow 1, M; y.N^0, y.N^1)$ is the *composite*, an x-line in A from $N^0\langle 1/y\rangle$ to $N^1\langle 1/y\rangle$. Moreover, there is an (x,y)-square with those four lines as its boundary, namely $\mathsf{hcom}^x_A(0 \rightsquigarrow y, M; y.N^0, y.N^1)$, which is called the *filler* of this composition scenario.

$$\begin{tikzcd}
\cdot \arrow[r, "x"] \arrow[d, "y"'] & {} \\
{} &
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdot \arrow[r, "M"] \arrow[d, "N^0"'] & \cdot \arrow[d, "N^1"] \\
N^0\langle 1/y\rangle \arrow[r, dashed, "\mathsf{hcom}^x_A(0 \rightsquigarrow 1, M; y.N^0, y.N^1)"'] & N^1\langle 1/y\rangle
\end{tikzcd}$$



The general form of hcom is $\mathsf{hcom}^{\overrightarrow{r_i}}_A(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon})$. M is the *cap* of the composition; r specifies which "side" the cap is on (above, the $y = 0$ side), and r' the side the composite is on (above, the $y = 1$ side). When r' is a dimension name not occurring elsewhere in the composition problem, the hcom "traces out" the interior of the composition problem, ranging from the cap to the composite, thus obtaining the filler. Finally, $\overrightarrow{r_i} = r_1, \dots, r_n$ is a list of $n \geq 1$ dimension terms called the *extents* of the composition problem, and $\overrightarrow{y.N_i^\varepsilon}$ is a list of $2n$ *tube faces*, each with a bound dimension name. Each extent specifies the dimension across which each pair of tube faces lies; for example, the first two tube faces are on the $r_1 = 0$ and $r_1 = 1$ sides of the composition problem (above, $x = 0$ and $x = 1$). The bound dimension names in the tube faces are the dimension in which r, r' lie; binding it ensures the cap and composite do not vary in that direction.

Above (right) is an example of a two-extent composition problem. If the top (x,z)-square is M, the left and right (y,z)-squares (across the x direction) are N^0_x, N^1_x, and the back and front (x,y)-squares (across the z direction) are N^0_z, N^1_z, then the shaded bottom face is the composite, an (x,z)-square $\mathsf{hcom}^{x,z}_A(0 \rightsquigarrow 1, M; y.N^0_x, y.N^1_x, y.N^0_z, y.N^1_z)$, and the interior (x,y,z)-cube filler is $\mathsf{hcom}^{x,z}_A(0 \rightsquigarrow y, M; y.N^0_x, y.N^1_x, y.N^0_z, y.N^1_z)$.

We say that A type $[\Psi]$ when A $\mathsf{pretype}$ $[\Psi]$ is Kan, that is, supports Kan composition and coercion:

$$\frac{\begin{array}{ll} & M \in A\ [\Psi] \\ (\forall i,j,\varepsilon,\varepsilon') & N_i^\varepsilon \doteq N_j^{\varepsilon'} \in A\ [\Psi, y \mid r_i = \varepsilon, r_j = \varepsilon'] \\ (\forall i,\varepsilon) & N_i^\varepsilon\langle r/y\rangle \doteq M \in A\ [\Psi \mid r_i = \varepsilon] \end{array}}{\mathsf{hcom}^{\overrightarrow{r_i}}_A(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}) \in A\ [\Psi]}
\qquad
\frac{M \in A\langle r/x\rangle\ [\Psi]}{\mathsf{coe}^{r \rightsquigarrow r'}_{x.A}(M) \in A\langle r'/x\rangle\ [\Psi]}$$

(See Angiuli and Harper [6, Definition 17] for the full statement of the Kan conditions, including additional laws regarding the faces of hcom and coe.) The above rules use *dimension context restrictions* to compactly state the preconditions of the Kan operation across all possible configurations. We say that a set of dimension term equations $\Xi = \{r_i = r_i'\}_i$ in Ψ is *satisfied* by $\psi : \Psi' \to \Psi$ if every $r_i\psi = r_i'\psi$. Then we say $M \doteq N \in A\ [\Psi \mid \Xi]$, presupposing $A\ \mathsf{pretype}\ [\Psi \mid \Xi]$, when for any $\psi : \Psi' \to \Psi$ satisfying Ξ, $M\psi \doteq N\psi \in A\psi\ [\Psi']$.

As before, the open judgments

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

express the truth of a judgment universally over members of $A_1, \dots, A_n$. The Ψ specifies at which dimension the context, pretype, and elements are initially considered, but the quantification over the context must hold at all aspects $A_1\psi, \dots, A_n\psi$. For example, $a_1 : A_1 \gg M \doteq M' \in B\ [\Psi]$, presupposing $a_1 : A_1 \gg B\ \mathsf{pretype}\ [\Psi]$, when for any $\psi : \Psi' \to \Psi$ and $N_1 \doteq N_1' \in A_1\psi\ [\Psi']$, we have $M\psi[N_1/a_1] \doteq M'\psi[N_1'/a_1] \in B\psi[N_1/a_1]\ [\Psi']$. We say $\Gamma \gg B \doteq B'\ \mathsf{type}\ [\Psi]$, presupposing $\Gamma \gg B \doteq B'\ \mathsf{pretype}\ [\Psi]$, when for any equal members of the context types, the corresponding instances of B, B' are equal closed types.

Since types are programs, and open judgments are given meaning by substitution, dependency is simply a consequence of allowing types to contain variables. It is natural that types can depend on dimension names, because if $a : A \gg B\ \mathsf{type}\ [\cdot]$ and M is an x-line of A, then $B[M/a]$ is a type varying in x, with endpoints $B[M\langle\varepsilon/x\rangle/a]$.

Finally, we define each type former, show the types satisfy the expected proof rules, and construct a cubical type system closed under those type formers. (See Angiuli and Harper [6] for the complete definitions and proofs.)

A cubical type system has the *dependent function type* of $A\ \mathsf{type}\ [\Psi]$ and $a : A \gg B\ \mathsf{type}\ [\Psi]$ if for all $\psi : \Psi' \to \Psi$, $(a{:}A\psi) \to B\psi$ is a canonical Ψ'-dimensional pretype, with canonical Ψ'-dimensional members $\lambda a.M$ whenever $a : A\psi \gg M \in B\psi\ [\Psi']$. Cubical versions of the ordinary rules for dependent function types hold in any cubical type system with all dependent function types:

$$\frac{a : A \gg M \in B\ [\Psi]}{\lambda a.M \in (a{:}A) \to B\ [\Psi]} \qquad \frac{M \in (a{:}A) \to B\ [\Psi] \quad N \in A\ [\Psi]}{\mathsf{app}(M, N) \in B[N/a]\ [\Psi]}$$

$$\frac{a : A \gg M \in B\ [\Psi] \quad N \in A\ [\Psi]}{\mathsf{app}(\lambda a.M, N) \doteq M[N/a] \in B[N/a]\ [\Psi]} \qquad \frac{M \in (a{:}A) \to B\ [\Psi]}{M \doteq \lambda a.\mathsf{app}(M, a) \in (a{:}A) \to B\ [\Psi]}$$

A cubical type system has the *circle type* if $\mathbb{S}^1$ is a canonical Ψ-dimensional pretype for all Ψ, with canonical Ψ-dimensional members base, loop_x, and $\mathsf{hcom}^{\overrightarrow{x_i}}_{\mathbb{S}^1}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon})$ whenever $r \neq r'$,

- $M \in \mathbb{S}^1\ [\Psi]$,
- $N_i^\varepsilon \doteq N_j^{\varepsilon'} \in \mathbb{S}^1\ [\Psi, y \mid x_i = \varepsilon, x_j = \varepsilon']$ for all $i, j, \varepsilon, \varepsilon'$, and
- $N_i^\varepsilon\langle r/y\rangle \doteq M \in \mathbb{S}^1\ [\Psi \mid x_i = \varepsilon]$ for all i, ε.

Because $\mathbb{S}^1$ is a base type, its Kan operations are freely added. (In contrast, the Kan operations of $(a{:}A) \to B$ can be implemented by the Kan operations of A and B, and are never canonical.) Recall that the Kan conditions force certain $\mathsf{hcom}_{\mathbb{S}^1}(\dots)$ to exist; some of these compute to a cap or tube face:

$$\frac{\overrightarrow{r_i} = x_1, \dots, x_{i-1}, \varepsilon, r_{i+1}, \dots, r_n}{\mathsf{hcom}^{\overrightarrow{r_i}}_{\mathbb{S}^1}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}) \longmapsto N_i^\varepsilon\langle r'/y\rangle} \qquad \frac{r = r'}{\mathsf{hcom}^{x_1,\dots,x_n}_{\mathbb{S}^1}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}) \longmapsto M}$$

while the others are freely added as canonical members of $\mathbb{S}^1$.

A cubical type system has *strict booleans* if sbool is a canonical Ψ-dimensional pretype for all Ψ, and its canonical Ψ-dimensional members are true and false. Because this type has no path constructors, its Kan operators admit a trivial implementation in which compositions of true are true and compositions of false are false:

$$\frac{}{\mathsf{hcom}^{\overrightarrow{r_i}}_{\mathsf{sbool}}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}) \longmapsto M}$$

We also define a type bool of booleans with free Kan operations, whose canonical elements are true, false, and $\mathsf{hcom}_{\mathsf{bool}}(\dots)$, following the definition of $\mathbb{S}^1$.

Given A type $[\Psi]$, B type $[\Psi \mid x = 1]$, and a strict isomorphism F, G between $A\langle 1/x\rangle$ and $B\langle 1/x\rangle$, that is:

- $F \in A \to B\ [\Psi \mid x = 1]$,
- $G \in B \to A\ [\Psi \mid x = 1]$,
- $a : A \gg \mathsf{app}(G, \mathsf{app}(F, a)) \doteq a \in A\ [\Psi \mid x = 1]$, and
- $b : B \gg \mathsf{app}(F, \mathsf{app}(G, b)) \doteq b \in B\ [\Psi \mid x = 1]$,

we say a cubical type system has their *strict univalence* type $\mathsf{ia}_x(A, B, F, G)$ when for any $\psi : \Psi' \to \Psi$ for which $x\psi = x'$, $\mathsf{ia}_{x'}(A\psi, B\psi, F\psi, G\psi)$ is a canonical Ψ'-dimensional pretype, with canonical Ψ'-dimensional members $\mathsf{iain}_x(M, F\psi)$ whenever $M \in A\psi\ [\Psi']$.

The type $\mathsf{ia}_x(A, B, F, G)$ constructs an x-line between $A\langle 0/x\rangle$ and $B\langle 1/x\rangle$ whose elements arise from elements of A. This is an instance of the univalence axiom, which states that every equivalence between types yields a line between them.

$$\begin{tikzcd}
\rightarrow x & A\langle 0/x\rangle \arrow[r, "A"] \arrow[dr, "\mathsf{ia}_x(A,B,F,G)"'] & A\langle 1/x\rangle \arrow[d, bend left, "F\langle 1/x\rangle"] \\
& & B\langle 1/x\rangle \arrow[u, bend left, "G\langle 1/x\rangle"]
\end{tikzcd}$$

If $M \in \mathsf{ia}_x(A, B, F, G)\ [\Psi]$, $M\langle 0/x\rangle$ must be an element of $A\langle 0/x\rangle$ and $M\langle 1/x\rangle$ must be an element of $B\langle 1/x\rangle$. In the operational semantics, taking the $\langle 1/x\rangle$ face of

$\mathsf{iain}_x(M, F)$ applies F:

$$\frac{}{\mathsf{iain}_x(M,F)\ \mathsf{val}} \qquad \frac{}{\mathsf{iain}_0(M,F) \longmapsto M} \qquad \frac{}{\mathsf{iain}_1(M,F) \longmapsto \mathsf{app}(F,M)}$$

The Kan operations of $\mathsf{ia}_x(A, B, F, G)$ are somewhat complicated, and rely on the Kan operations of A and B as well as both directions of the isomorphism.

In addition to dependent function, circle, boolean, strict boolean, and strict univalence types, Angiuli and Harper [6] define dependent pair types, path types (which abstract a dimension name), and not_x (a special case of strict univalence). We construct a cubical type system closed under these type formers using a fixed point construction [1, 25] that starts with an empty cubical type system, and adjoins the base types and new higher types at each step.

We define a complete partial order on cubical type systems (a partial order with a least element and joins of all directed subsets) with the approximation ordering $\tau \sqsubseteq \tau'$, which holds (roughly) when all types in τ are types in τ' with precisely the same canonical members and equality. Whenever $\tau \sqsubseteq \tau'$, any judgment $\mathcal{J}\ [\Psi]$ that holds in τ (which we write $\tau \models (\mathcal{J}\ [\Psi])$) also holds in τ' ($\tau' \models (\mathcal{J}\ [\Psi])$).

We define a monotone operator $F(E, \Phi) = (E', \Phi')$ on cubical type systems which adds one "layer" of types to (E, Φ):

$$\begin{aligned}
E' = \{&(\Psi, \mathbb{S}^1, \mathbb{S}^1)\} \cup \dots \\
&\cup \{(\Psi, (a{:}A) \to B, (a{:}A') \to B') \mid \\
&\qquad (E,\Phi) \models (A \doteq A'\ \mathsf{type}\ [\Psi]) \wedge (E,\Phi) \models (a : A \gg B \doteq B'\ \mathsf{type}\ [\Psi])\} \\
\Phi' = \{&(\Psi, \mathbb{S}^1, V, V') \mid \mathbb{C}(\Psi, V, V')\} \cup \dots \\
&\cup \{(\Psi, (a{:}A) \to B, \lambda a.M, \lambda a.M') \mid (E,\Phi) \models (a : A \gg M \doteq M' \in B\ [\Psi])\}
\end{aligned}$$

(Here, $\mathbb{C}$ is an inductively-defined family of PERs generated by base, loop_x, and canonical $\mathsf{hcom}_{\mathbb{S}^1}(\dots)$ elements.) It is easy to check that F is monotone, and that any fixed point of F is closed under the type formers. Monotone operators on complete partial orders (CPOs) have least fixed points [21, 8.22], completing the construction.

Note, however, that *any* cubical type system closed under the type formers is sufficient for our purposes; none of our theorems hold only in the least such. It is therefore possible to extend our results with additional type formers (such as universes, full univalence, or more higher inductive types) without affecting the present constructions.

Computational higher-dimensional type theory (with the exception of strict univalence) is implemented in the experimental RedPRL proof assistant [54]. Angiuli et al. [8] proved the first canonicity result for a fully higher-dimensional type theory, using a non-dependent version of the cubical meaning explanations. Subsequently, Huber [33] proved canonicity for the formal higher-dimensional type theory of Cohen et al.

[16]. (Several years prior, Licata and Harper [39] obtained a canonicity result for a higher-dimensional type theory with only points and lines.)

Our Kan conditions are very similar to those considered by Licata and Brunerie [38], which were inspired by the uniform Kan condition of Bezem et al. [13]. Other variations on the Kan conditions are possible: Cohen et al. [16] consolidate hcom and coe into a single operator, and only allow computing the $\langle 1/y\rangle$ face of a filler given its $\langle 0/y\rangle$ face and any configuration of faces with extent in the y direction.

Other variations on the dimension terms are also possible, yielding different operations on cubes: while we allow only faces, diagonals, and degeneracies, Cohen et al. [16] also allow *reversals* ($\langle (1-x)/x\rangle$) and *connections* ($\langle (x \wedge y)/x\rangle$ and $\langle (x \vee y)/x\rangle$), and rely on connections to define Kan fillers from their notion of Kan composition.

3 Proposed Work

I intend to pursue a variety of extensions and applications of computational higher-dimensional type theory, falling into four main categories:

- Extensions inspired by other higher-dimensional type theories—universes (Section 3.1), univalence (Section 3.2), and equality pretypes (Section 3.3);
- Extensions inspired by other computational type theories—exact quotients, computational equivalence, and extensional and untyped reasoning principles (Section 3.3);
- Programming applications—generic programming and proof-relevant quotients (Section 3.4); and
- Connections to related work—book HoTT, Cohen et al. [16], and cubical sets (Section 3.5).

3.1 Universes

Universes in dependent type theory internalize the notion of typehood, allowing one to speak directly about, for instance, pairs of types, or indexed families of types. Universes also allow users to construct types by induction; without universes (or large elimination principles), formal type theories cannot prove injectivity of constructors [52]. In book HoTT, the statement of the univalence axiom requires universes, as univalence places a condition on paths between types ($A =_{\mathcal{U}} B$). The present setting allows for multiple notions of universes, which I describe in turn.

A primitive form of universe can be constructed as follows. Extend the term language with a new value $\mathcal{U}$. Let $\tau_0 = (E_0, \Phi_0)$ be the least fixed point of the monotone operator F described in Section 2.4. We consider a new monotone operator on cubical type systems $F_1(E, \Phi) = (E', \Phi')$ defined like F, extended with the clauses:

$$\begin{aligned} E' &= \cdots \cup \{(\Psi, \mathcal{U}, \mathcal{U})\} \\ \Phi' &= \cdots \cup \{(\Psi, \mathcal{U}, A, B) \mid E_0(\Psi, A, B)\} \end{aligned}$$

Let τ_1 be the least fixed point of F_1. One can verify that $F(\tau) \sqsubseteq F_1(\tau)$ for all τ, and thus $\tau_0 \sqsubseteq \tau_1$. The type system τ_1 satisfies all the proof rules described earlier, plus:

$$\frac{}{\mathcal{U}\ \mathsf{pretype}\ [\Psi]} \qquad \frac{A \in \mathcal{U}\ [\Psi]}{A\ \mathsf{pretype}\ [\Psi]}$$

$$\frac{}{\mathbb{S}^1 \in \mathcal{U}\ [\Psi]} \qquad \frac{A \in \mathcal{U}\ [\Psi] \quad a : A \gg B \in \mathcal{U}\ [\Psi]}{(a{:}A) \to B \in \mathcal{U}\ [\Psi]} \qquad \cdots$$

One obtains a hierarchy of universes by considering a sequence of monotone operators F_i, each defining $\mathcal{U}_j$ for all $j < i$ as the PER of canonical pretypes in the least fixed point of F_j.[5] The resulting $\mathcal{U}_j$ are universes à la Russell (elements of $\mathcal{U}$ are precisely pretypes) rather than à la Tarski as in most formal type theories (requiring a function $\mathsf{El}(-)$ sending elements of $\mathcal{U}$ to types). The latter is more natural in categorical semantics [42], but the former is straightforward in computational type theory, as it lacks any *a priori* distinction between types and terms.

The above argument is a cubical generalization of the universe construction given by Harper [25], and appears to rely on non-constructive facts about CPOs. Allen [1] gives an intuitionistically-acceptable construction which avoids CPOs, employing an operator on bare relations rather than on type systems. It is not clear that Allen's construction generalizes to cubical type systems, as the definitions of the judgments in terms of type systems are rather more involved in cubical type theory than in ordinary type theory; I wish to investigate this point.

The construction outlined above yields a pretype $\mathcal{U}$ that is not Kan. One cannot eliminate from $\mathbb{S}^1$ into such a $\mathcal{U}$, as the elimination principle for $\mathbb{S}^1$ sends $\mathsf{hcom}_{\mathbb{S}^1}$ to a composition in the result type. Because $\mathcal{U}$ is a base type, it can be equipped with a free Kan structure in much the same way as bool and $\mathbb{S}^1$. First, equip $\mathsf{hcom}_{\mathcal{U}}(\dots)$ terms with the minimal operational semantics needed for the Kan conditions:

$$\frac{\overrightarrow{r_i} = x_1, \dots, x_{i-1}, \varepsilon, r_{i+1}, \dots, r_n}{\mathsf{hcom}_{\mathcal{U}}^{\overrightarrow{r_i}}(r \rightsquigarrow r', A; \overrightarrow{y.B_i^\varepsilon}) \longmapsto B_i^\varepsilon\langle r'/y\rangle} \qquad \frac{r = r'}{\mathsf{hcom}_{\mathcal{U}}^{x_1,\dots,x_n}(r \rightsquigarrow r', A; \overrightarrow{y.B_i^\varepsilon}) \longmapsto A}$$

$$\frac{r \neq r'}{\mathsf{hcom}_{\mathcal{U}}^{x_1,\dots,x_n}(r \rightsquigarrow r', A; \overrightarrow{y.B_i^\varepsilon})\ \mathsf{val}} \qquad \frac{}{\mathsf{coe}_{\mathcal{U}}^{r\rightsquigarrow r'}(A) \longmapsto A}$$

Then, expand the canonical elements of $\mathcal{U}$ to include well-typed canonical $\mathsf{hcom}_{\mathcal{U}}(\dots)$ elements, that is, $\mathsf{hcom}_{\mathcal{U}}^{\overrightarrow{x_i}}(r \rightsquigarrow r', A; \overrightarrow{y.B_i^\varepsilon}) \approx_{\mathcal{U}}^{\Psi} \mathsf{hcom}_{\mathcal{U}}^{\overrightarrow{x_i}}(r \rightsquigarrow r', C; \overrightarrow{y.D_i^\varepsilon})$ when $r \neq r'$,

- $\tau_0 \models (A \doteq C\ \mathsf{type}\ [\Psi])$,
- $\tau_0 \models \big(B_i^\varepsilon \doteq B_j^{\varepsilon'}\ \mathsf{type}\ [\Psi, y \mid x_i = \varepsilon, x_j = \varepsilon']\big)$ for all $i, j, \varepsilon, \varepsilon'$,
- $\tau_0 \models (B_i^\varepsilon \doteq D_i^\varepsilon\ \mathsf{type}\ [\Psi, y \mid x_i = \varepsilon])$ for all i, ε, and
- $\tau_0 \models (B_i^\varepsilon\langle r/y\rangle \doteq A\ \mathsf{type}\ [\Psi \mid x_i = \varepsilon])$ for all i, ε.

If this modified $\mathcal{U}$ is to remain a universe, all of its elements—including all $\mathsf{hcom}_{\mathcal{U}}(\dots)$—must be types, that is, they must themselves have elements. The Kan equations demand that these elements have certain properties. In the scenario:

[5] It transpires that no $\mathcal{U}_i$ is an element of itself, but for reasons of predicativity, rather than size; F_1 would not be monotone if it assigned $\mathcal{U}_0$ the PER E rather than E_0, as the type $\mathcal{U}_0$ would grow larger in a larger type system.

$$\begin{array}{c} A \ \mathsf{type}\ [\Psi, y] \\ B \ \mathsf{type}\ [\Psi, x] \\ C \ \mathsf{type}\ [\Psi, y] \\ A\langle 0/y\rangle \doteq B\langle 0/x\rangle \ \mathsf{type}\ [\Psi] \\ C\langle 0/y\rangle \doteq B\langle 1/x\rangle \ \mathsf{type}\ [\Psi] \end{array}$$

$$\begin{tikzcd}
{} \arrow[r, "x"] \arrow[d, "y"'] & {} & \bullet \arrow[rr, "B"] \arrow[d, "A"'] \arrow[drr, phantom, "F := \mathsf{hcom}^x_{\mathcal{U}}(0 \rightsquigarrow y, B; y.A, y.C)"] & & \bullet \arrow[d, "C"] \\
{} & & \bullet \arrow[rr, "\mathsf{hcom}^x_{\mathcal{U}}(0 \rightsquigarrow 1, B; y.A, y.C)"'] & & \bullet
\end{tikzcd}$$

the Kan conditions for $\mathcal{U}$ require $F\langle 0/x\rangle \doteq A \in \mathcal{U}\ [\Psi, y]$, and hence, that $F\langle 0/x\rangle \doteq A\ \mathsf{pretype}\ [\Psi, y]$. As a result, any element $X \in F\ [\Psi, x, y]$ must have faces that are elements of A ($X\langle 0/x\rangle \in A\ [\Psi, y]$) and of the composite ($X\langle 1/y\rangle \in F\langle 1/y\rangle\ [\Psi, x]$). A straightforward way to ensure these properties is to define the canonical elements of every canonical $\mathsf{hcom}_{\mathcal{U}}(\dots)$ as *formal composites* of elements of the constituent types, of the same shape. These are given the same operational semantics as $\mathsf{hcom}_{\mathcal{U}}(\dots)$:

$$\frac{\overrightarrow{r_i} = x_1, \dots, x_{i-1}, \varepsilon, r_{i+1}, \dots, r_n}{\mathsf{box}^{\overrightarrow{r_i}}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}) \longmapsto N_i^\varepsilon\langle r'/y\rangle} \qquad \frac{r = r'}{\mathsf{box}^{x_1,\dots,x_n}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}) \longmapsto M}$$

$$\frac{r \neq r'}{\mathsf{box}^{x_1,\dots,x_n}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon})\ \mathsf{val}}$$

In our running example, we would say that $\mathsf{box}^x(0 \rightsquigarrow y, N; y.M, y.P)$ is a canonical element of F whenever:

$$\begin{array}{c} M \in A\ [\Psi, y] \\ N \in B\ [\Psi, x] \\ P \in C\ [\Psi, y] \\ M\langle 0/y\rangle \doteq N\langle 0/x\rangle \in B\langle 0/x\rangle\ [\Psi] \\ P\langle 0/y\rangle \doteq N\langle 1/x\rangle \in B\langle 1/x\rangle\ [\Psi] \end{array}$$

As required, $-\langle 1/y\rangle$ of this composite steps to P, an element of C. Extend the original fixed point construction with $\mathsf{hcom}_{\mathcal{U}}$ pretypes whose canonical elements are boxes; then $\mathcal{U}$, as previously constructed, will be a *type* whose elements are pretypes.

Many constructions in book HoTT require a universe type whose elements are also types. To compute the fundamental group of the circle, one must both construct a map from $\mathbb{S}^1$ into $\mathcal{U}$ (requiring $\mathcal{U}$ to be Kan) and coerce along the image of that map (requiring the elements of $\mathcal{U}$ to be Kan) [40].

We must therefore define the canonical elements of $\mathcal{U}$ not as canonical pretypes in τ_0 but rather canonical types:

$$\Phi_1(\Psi, \mathcal{U}, A, B) \iff (E_0, \Phi_0) \models (A \doteq B\ \mathsf{type}\ [\Psi])$$

As I will discuss in Section 3.3, I want to allow for the possibility of pretypes that are not also types, although none are currently defined in Angiuli and Harper [6].

The above definition exposes a mismatch in the current definition of cubical type systems—although we can only specify the canonical elements of $\mathcal{U}$, the condition of being a type is defined for arbitrary terms, and it is not necessarily the case that whenever A is a type, its value $A \Downarrow A_0$ is also a type. Such a condition can be explicitly added, and is analogous to the cubicality condition already required of elements.

For such a $\mathcal{U}$ to be Kan, however, the types $\mathsf{hcom}_{\mathcal{U}}(\dots)$ must themselves be Kan. That is, we must implement the operations $\mathsf{hcom}_{\mathsf{hcom}_{\mathcal{U}}(\dots)}(\dots)$ and $\mathsf{coe}_{w.\mathsf{hcom}_{\mathcal{U}}(\dots)}(\dots)$. In our running example, given any composable box of formal boxes in F, we must produce a formal box in F; and given any element of F or one of its faces, we must produce an element of F or its opposite face. Doing so requires extending the operational semantics and proving that the given operations are defined coherently. This task is quite challenging, especially because these operations do not always take boxes to boxes; consider $\mathsf{coe}^{0\rightsquigarrow y}_{y.F}(M)$, which must take elements of B to elements of F, in such a way that $-\langle 0/x\rangle$ of the result is equal to $\mathsf{coe}^{0\rightsquigarrow y}_{y.A}(M\langle 0/x\rangle)$.

I consider the definition of $\mathsf{hcom}_{\mathsf{hcom}_{\mathcal{U}}(\dots)}(\dots)$ and $\mathsf{coe}_{w.\mathsf{hcom}_{\mathcal{U}}(\dots)}(\dots)$ to be the most challenging and risky task in this proposal. It is quite possible that defining such operations requires modifying how the elements of $\mathsf{hcom}_{\mathcal{U}}(\dots)$ are defined, or even modifying the Kan operations or cube operations of the entire type theory. Despite the importance of a Kan universe of Kan types for synthetic homotopy theory, I have intentionally chosen other tasks in my proposal that do not rely on such a universe, and I view these tasks as sufficient evidence for my thesis statement in the event that I cannot define such a universe in a timely fashion.

As a point of comparison, the formal cubical type theory of Cohen et al. [16] defines a Kan, univalent universe of Kan types, using the *gluing construction* to simultaneously define the analogue of $\mathsf{hcom}_{\mathcal{U}}(\dots)$ types and univalence types (Section 3.2). Their gluing construction relies crucially on their Kan operation being limited to compositions $0 \rightsquigarrow 1$, eliminating the need for equations such as those relating F with A and C in our running example. (In exchange, connections and reversals are needed in their setting to define Kan fillers.)

While the gluing construction does not seem to apply in my setting, I am nevertheless encouraged that a similar problem has been solved. Even if a univalent cubical universe is *only* definable using the precise Kan operations and cube operations considered by Cohen et al. [16]—which would be quite unexpected—I believe it would be possible to adapt such operations to my present setting, as the cubical meaning explanations do not rely in an essential way on the details of those operations.

3.2 Univalence

The univalence axiom of book HoTT postulates an equivalence between paths in the universe and equivalences between types. That equivalence induces paths in the universe between all pairs of equivalent types, coercion along which applies the equivalence between those types (up to a path).

To validate the univalence axiom in this setting, there must be a line from A to B for any types A, B and an equivalence E between them. By analogy with the types not_x and $\mathsf{ia}_x(A, B, F, G)$, we say there is a line $\mathsf{ua}_x(A, B, E)$ whenever:

$$\begin{gathered} A\ \mathsf{type}\ [\Psi] \\ B\ \mathsf{type}\ [\Psi \mid x = 1] \\ E \in \mathsf{Equiv}(B, A)\ [\Psi \mid x = 1] \end{gathered}$$

$$\begin{tikzcd} \rightarrow x & A\langle 0/x\rangle \arrow[r, "A"] \arrow[dr, "\mathsf{ua}_x(A,B,E)"'] & A\langle 1/x\rangle \\ & & B\langle 1/x\rangle \arrow[u, "\mathsf{fst}(E\langle 1/x\rangle)"'] \end{tikzcd}$$

We define $\mathsf{Equiv}(B, A)$ as in the HoTT Book [58, Definition 4.2.1]. Recall that any equivalence $E \in \mathsf{Equiv}(B, A)\ [\Psi]$ contains a function $\mathsf{fst}(E) \in B \to A\ [\Psi]$ and additional data from which a weak inverse $E^{-1} \in A \to B\ [\Psi]$ can be computed; the compositions of these functions differ by a path from the identity function in both directions.

A *univalent universe* is simply a universe with a path for each equivalence, satisfying the additional property that the type of paths is equivalent to the type of equivalences. Dan Licata has shown (in the setting of book HoTT) that the univalence axiom holds in any Kan universe of Kan types in which paths exist for every equivalence, and in which there is a path between $\mathsf{coe}^{0\rightsquigarrow 1}_{x.\mathsf{ua}_x(A,B,E)}(M)$ and $\mathsf{app}(E^{-1}, M)$ [36].

In order to be an element of such a universe, $\mathsf{ua}_x(A, B, E)$ must itself be Kan. The Kan operations on $\mathsf{ua}_x(A, B, E)$ force it to have elements, because $\mathsf{coe}^{0\rightsquigarrow x}_{x.\mathsf{ua}_x(A,B,E)}(M)$ computes an element of it when $M \in A\langle 0/x\rangle\ [\Psi]$. Following the definition of $\mathsf{ia}_x(A, B, F, G)$, we say that an element of $\mathsf{ua}_x(A, B, E)$ is given by an element of A (with constructor $\mathsf{uain}_x(-)$), and implement the Kan operations of $\mathsf{ua}_x(A, B, E)$ using the Kan operations of A. Such implementations are simplified by defining an eliminator $\mathsf{uaout}_x(-)$ to project out the underlying element of A.

We also need to ensure that whenever $M \in \mathsf{ua}_x(A, B, E)\ [\Psi, x]$, $M\langle 0/x\rangle$ is an element of $A\langle 0/x\rangle$ and $M\langle 1/x\rangle$ is an element of $B\langle 1/x\rangle$. (We satisfy the latter requirement by explicitly providing an element of $B\langle 1/x\rangle$ in the introduction form.) This complicates the definition of $\mathsf{uaout}_x(-)$, as under $\langle 1/x\rangle$ it must compute an element of $A\langle 1/x\rangle$ from an element of $B\langle 1/x\rangle$ (presumably, by applying $\mathsf{fst}(E\langle 1/x\rangle)$). The $\mathsf{uaout}_x(-)$ operation is coherent only if the provided element of $B\langle 1/x\rangle$ is sent by $\mathsf{fst}(E\langle 1/x\rangle)$ to exactly the $\langle 1/x\rangle$ face of the provided element of A. These considerations can be summarized by the following intended introduction and elimination rules:

$$\frac{M \in A\ [\Psi] \quad N \in B\ [\Psi \mid r=1] \quad \mathsf{app}(\mathsf{fst}(E), N) \doteq M \in A\ [\Psi \mid r=1]}{\mathsf{uain}_r(M, N) \in \mathsf{ua}_r(A, B, E)\ [\Psi]}$$

$$\frac{M \in \mathsf{ua}_r(A, B, E)\ [\Psi] \quad \mathsf{fst}(E) \doteq F \in B \to A\ [\Psi \mid r=1]}{\mathsf{uaout}_r(M, F) \in A\ [\Psi]}$$

as well as the following operational semantics:

$$\frac{}{\mathsf{ua}_x(A, B, E)\ \mathsf{val}} \qquad \frac{}{\mathsf{ua}_0(A, B, E) \longmapsto A} \qquad \frac{}{\mathsf{ua}_1(A, B, E) \longmapsto B}$$

$$\frac{}{\mathsf{uain}_x(M, N)\ \mathsf{val}} \qquad \frac{}{\mathsf{uain}_0(M, N) \longmapsto M} \qquad \frac{}{\mathsf{uain}_1(M, N) \longmapsto N}$$

$$\frac{}{\mathsf{uaout}_0(M, F) \longmapsto M} \qquad \frac{}{\mathsf{uaout}_1(M, F) \longmapsto \mathsf{app}(F, M)}$$

$$\frac{M \longmapsto M'}{\mathsf{uaout}_x(M, F) \longmapsto \mathsf{uaout}_x(M', F)} \qquad \frac{}{\mathsf{uaout}_x(\mathsf{uain}_x(M, N), F) \longmapsto M}$$

It remains to define the Kan operations in a coherent fashion. For instance, the implementation of $\mathsf{hcom}^{\overrightarrow{r_i}}_{\mathsf{ua}_x(A,B,E)}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon})$ must have a $\langle 0/x \rangle$ face equal to the composition $(\mathsf{hcom}^{\overrightarrow{r_i}}_A(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}))\langle 0/x \rangle$ and a $\langle 1/x \rangle$ face equal to the composition $(\mathsf{hcom}^{\overrightarrow{r_i}}_B(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}))\langle 1/x \rangle$. Similar challenges already arise in the Kan operations of $\mathsf{ia}_x(A, B, F, G)$, but $\mathsf{ua}_x(A, B, E)$ has the additional difficulty that we must provide an exact preimage of the $\langle 1/x \rangle$ face of the composition in A, even though $\mathsf{fst}(E\langle 1/x \rangle)$ has no exact inverse.

When combined with the work described in Section 3.1, the above construction equips computational higher-dimensional type theory with a univalent, Kan universe of Kan types, making it a suitable framework for synthetic homotopy theory in the style of the HoTT Book [58]. In Section 3.4, I argue that even in the absence of a universe, Kan $\mathsf{ia}_x(A, B, F, G)$ and $\mathsf{ua}_x(A, B, E)$ types have interesting programming applications.

The construction of $\mathsf{ia}_x(A, B, F, G)$, described in Section 2.4, is possible in our setting because we have an extensional judgmental equality distinct from the path type. As a result, we can easily consider universes with both kinds of univalence—for strict isomorphisms as well as equivalences. An interesting question is: how do these two types compare? Is there a "strict univalence axiom" for $\mathsf{ia}_x(A, B, F, G)$ analogous to the ordinary univalence axiom? The implementations of the Kan operations of $\mathsf{ia}_x(A, B, F, G)$ will almost certainly be simpler than of $\mathsf{ua}_x(A, B, E)$; in the case where an equivalence is also a strict isomorphism, is the former more efficient (say, in the size of the term, or number of steps required)?

3.3 Extensional Principles

Most formal higher-dimensional type theories, including book HoTT and Cohen et al. [16], cannot express exact, extensional equality of open terms. In such theories, one has a path between two functions $A \to B$ whenever they agree on every element of A up to a path in B, but maintaining families of such paths can result in infinite towers of coherence data, and is believed to be an obstacle to defining semi-simplicial types in book HoTT [3]. Consequently, various researchers have proposed "two-level" type theories in which one can manipulate exact equalities in addition to paths [3, 62].

Furthermore, as remarked in Angiuli et al. [9], it is clear that behavioral equality of higher-dimensional programs is captured neither by definitional equality, which is not extensional, nor by the identity type, as $\mathsf{0}_{\mathbb{I}}$ and $\mathsf{1}_{\mathbb{I}}$ are distinct observations in $\mathbb{I}$ despite being connected by a path.

The equality judgment of the cubical meaning explanations is naturally an exact, extensional equality, because open terms are considered extensionally as maps from contexts to types. I have already relied on these semantics to validate inherently extensional equalities, even between higher inductive types, as in the proof that $\mathsf{not}(\mathsf{not}(M)) \doteq M \in \mathsf{bool}\ [\Psi]$ whenever $M \in \mathsf{bool}\ [\Psi]$ [6, Lemma 40].

In fact, one can already define a pretype internalizing the equality judgment:

$$\frac{\begin{array}{c} A \doteq A' \ \mathsf{pretype}\ [\Psi] \\ M \doteq M' \in A\ [\Psi] \\ N \doteq N' \in A\ [\Psi] \end{array}}{(M \doteq N \in A) \doteq (M' \doteq N' \in A')\ \mathsf{pretype}\ [\Psi]}$$

$$\frac{M \doteq N \in A\ [\Psi]}{\star \in (M \doteq N \in A)\ [\Psi]} \qquad \frac{P \in (M \doteq N \in A)\ [\Psi]}{M \doteq N \in A\ [\Psi]}$$

whose canonical elements are $\star \approx^{\Psi}_{(M \doteq N \in A)} \star$ whenever $M \doteq N \in A\ [\Psi]$. This pretype even has $\mathsf{hcom}_{(M \doteq N \in A)}(\dots)$, defined analogously to $\mathsf{hcom}_{\mathsf{sbool}}(\dots)$. Unfortunately, one cannot define $\mathsf{coe}_{x.(M \doteq N \in A)}(\dots)$ for the obvious reason that exact equality does not respect paths: if P is an x-line from M to M', then $\mathsf{coe}^{0 \rightsquigarrow 1}_{x.(M \doteq N \in A)}(\star)$ would send a proof that $M \doteq N \in A\ [\Psi]$ to a proof that $M' \doteq N \in A\ [\Psi]$. ("Two-level" type theories handle this problem essentially by axiomatizing both Kan types and non-Kan types.)

In NuPRL, the equality type is used to inductively prove extensional equalities; the same should hold in computational higher-dimensional type theory for sbool (and any future strict types), although one must prove a more general form of its elimination principle. Such equations would enable us to avoid the coherence towers described above, and are also needed to define interesting instances of $\mathsf{ia}_x(A, B, F, G)$. I will pursue exposing these reasoning principles in a two-level-style proof theory.

The trick of eliminating into an equality type to prove equations does not work for higher inductive types, because they can only eliminate into Kan types. As a result, we have no proof rule for showing $\mathsf{not}(\mathsf{not}(M)) \doteq M \in \mathsf{bool}\ [\Psi]$. Can we formulate a different principle for proving such equations? Such a principle would have to reason about canonical $\mathsf{hcom}_{\mathsf{bool}}$ terms in addition to true and false.

Because they are based on an untyped operational semantics, the cubical meaning explanations also justify more exotic reasoning principles. In the technical development, I frequently use the *head expansion* lemma that if $M' \doteq N \in A\ [\Psi]$ and for all $\psi : \Psi' \to \Psi$, $M\psi \longmapsto^* M'\psi$, then $M \doteq N \in A\ [\Psi]$ [6, Lemma 24]. Such a principle is helpful because most operational semantics steps, including ordinary β-reductions, are unaffected by dimension substitutions. Head expansion itself is not directly suitable for use in a proof assistant, because it does not apply to open terms, and requires reasoning explicitly about the operational semantics. However, it should be possible to prove a rule for open terms:

$$\frac{M' \doteq N \in A\ [\Psi] \quad M \to M'}{M \doteq N \in A\ [\Psi]}$$

for an explicitly-given open reduction relation $M \to M'$ that is stable under both dimension and term substitutions, and includes ordinary β-reductions.

The cubical meaning explanations coincide at dimension zero with the original NuPRL semantics [1, 25], so it should be possible to import many NuPRL type formers into the setting of higher-dimensional type theory, including (exact) quotients of types by equivalence relations. Such constructions would be useful even if not Kan, in the presence of a two-level-style proof theory allowing pretype reasoning. Ideally one could define cubical generalizations of these constructions, valid at arbitrary dimension. It seems plausible that one could quotient a cubical set by some form of cubical equivalence relation, but what rules are required?

More modern NuPRL semantics do not require types to be defined by their canonical elements; instead, all terms are considered modulo untyped computational equivalence [31], and types are any PERs respecting this equivalence. New types definable in this setting include the base type of *all* terms modulo equivalence; *partial* types of possibly non-terminating computations [53]; and PER types, which, along with base, suffice to define most other types [4]. A natural question is: how does untyped computational equivalence generalize in the cubical setting?

Intriguingly, a pretype whose canonical elements are every canonical term (modulo, say, α-equivalence) does not have all terminating computations as elements, because not all terminating computations have coherent aspects. Can the cubical meaning explanations be modified to allow defining types as evaluation-respecting cubical PERs on terms, rather than cubical PERs on canonical terms? What is the proper notion of coherence of aspects in such a setting?

3.4 Programming Applications

Prior to my work on computational higher-dimensional type theory, I considered programming applications of higher inductive types and univalent universes. In Angiuli et al. [9], I model version control systems as higher inductive types, where repository states are points, patches are paths, and laws equating patch sequences are paths between paths. Maps from these types into the universe are implementations, sending repositories to types, and (equal) patches to (equal) equivalences. However, Angiuli et al. [9] takes place in book HoTT, and relies on a conjectural notion of computation-up-to-homotopy. If I define a Kan, univalent universe of Kan types, I hope to re-examine some of these examples in the present setting.

Univalence, and especially the $\mathsf{ia}_x(A, B, F, G)$ type, has programming applications even in the absence of higher inductive types or universes of Kan types. Let $\mathcal{U}$ be a (not necessarily Kan) universe of Kan types closed under $\mathsf{ia}_x(A, B, F, G)$ types, and define $\mathsf{Magma} := (A{:}\mathcal{U}) \to (A \times A \to A)$. If F, G is a strict isomorphism between A and B, then the program $\mathsf{coe}^{0\rightsquigarrow 1}_{x.\mathsf{app}(\mathsf{Magma},\mathsf{ia}_x(A,B,F,G))}(-)$ takes a $\mathsf{Magma}(A)$ to a $\mathsf{Magma}(B)$. While one could always write such a program manually, coe is able to automatically apply the isomorphism as necessary! If the universe is Kan, univalence can also generate representation independence results stating that isomorphic implementations of existential types can be exchanged automatically [37]. I intend to describe a number of such examples.

Higher inductive types also have applications in the absence of univalence and universes, as illustrated in the following example (written using pseudocode in the style of Angiuli et al. [9]). Consider a type of natural numbers with paths between all numbers:

```
space I* : Type where
  num : Nat → I*
  path n : Id (num n) (num n+1)
```

We can define addition on `I*` by sending paths to the appropriate paths:

```
add1 : I* → I*
add1 (num n) = num n+1
add1 (path n) = path n+1

sum : I* → I* → I*
sum r (num 0) = r
sum r (num n+1) = add1 (sum r (num n))
(sum r) (path n) = ?? : Id (sum r (num n)) (sum r (num n+1))
```

In the above code, `??` requires a routine side proof, omitted, of `Id s (add1 s)`.

The type `I*` is similar to a quotient of `Nat` by the total relation, in the sense that its points must be all sent to path-connected points. Unlike an exact quotient, however, its points are not considered judgmentally equal, and `add1` is not extensionally equal to the identity. Consider two functions `fib` and `ffib` which return the `n`th Fibonacci number paired with the number of recursive calls used, as an element of `I*`:

```
fib : Nat → Nat × I*
fib 0   = (1, num 1)
fib 1   = (1, num 1)
fib n+2 = let (a,wa) = fib n
              (b,wb) = fib n+1
          in (a+b, add1 (sum wa wb))

ffib : Nat → Nat × I*
ffib n = let ((a,b),w) = ffib' n
         in (a,w)

ffib' : Nat → (Nat × Nat) × I*
ffib' 0   = ((1,1), num 1)
ffib' n+1 = let ((a,b),w) = ffib' n
            in ((b,a+b),add1 w)
```

`fib` and `ffib` are related by a path: on all inputs, their first components agree exactly, and their second components, being elements of `I*`, are always related by a path. They are not, however, extensionally equal: `fib 3` is `(3,num 5)` while `ffib 3` is `(3,num 4)`. Thus `ffib` cannot be silently replaced by `fib`, but one can always be substituted for the other given a path—acting here as a kind of resource-aware annotation—to adjust the second component. I plan to find further programming applications of these "proof-relevant quotients."

The `fib`/`ffib` example requires canonicity at dimension zero for `I*`, lest points in `I*` not have the form `num n`. Although I have not yet defined `I*`, a similar result does hold for the higher inductive types bool and $\mathbb{S}^1$ (whose canonical points are true and false, and base, respectively). Interestingly, the type theory of Cohen et al. [16] does *not* have this property for points of higher inductive types; their canonicity result holds only for a strict form of $\mathbb{N}$ (similar to our result for sbool). Our sharper canonicity result is possible because the hcom and coe Kan operators are distinct.[6]

Nearly all interesting programming applications require inductive types (both ordinary and higher-dimensional) not currently considered in Angiuli and Harper [6]. Natural numbers, lists, coproducts, *et cetera*, would also provide more examples of strict isomorphisms and interesting extensional equalities. I intend to define several such inductive types, whose definitions I expect to follow the pattern of sbool.

Surprisingly, these definitions can take advantage of computation being untyped. Consider a strict coproduct $A + B$ whose canonical elements at every dimension are

[6]In Cohen et al. [16], $\mathsf{comp}^i\ S^1\ []\ \mathsf{base}$ is a canonical point in S^1. The eliminator of S^1 sends $\mathsf{comp}^i\ S^1\ [(j = 0) \mapsto t]\ \mathsf{base}$ to $\mathsf{comp}^i\ A\ [(j = 0) \mapsto a']\ a$. $\langle 1/j \rangle$ of the latter is $\mathsf{comp}^i\ A\ []\ a$, which is not equal to a; therefore $\langle 1/j \rangle$ of the former, $\mathsf{comp}^i\ S^1\ []\ \mathsf{base}$, cannot be base.

$\mathsf{inl}(-)$ and $\mathsf{inr}(-)$. A natural implementation of hcom_{A+B} might evaluate the cap and every tube face, then:

$$\overline{\mathsf{hcom}^{\overrightarrow{r_i}}_{A+B}(r \rightsquigarrow r', \mathsf{inl}(M); \overrightarrow{y.\mathsf{inl}(N_i^\varepsilon)}) \longmapsto \mathsf{inl}(\mathsf{hcom}^{\overrightarrow{r_i}}_{A}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}))}$$

While conceptually clean, such an implementation requires complex operational semantics and coherence conditions. Instead, one can evaluate *only* the cap, then:

$$\overline{\mathsf{hcom}^{\overrightarrow{r_i}}_{A+B}(r \rightsquigarrow r', \mathsf{inl}(M); \overrightarrow{y.N_i^\varepsilon}) \longmapsto \mathsf{inl}(\mathsf{hcom}^{\overrightarrow{r_i}}_{A}(r \rightsquigarrow r', M; \overrightarrow{y.\mathsf{case}(N_i^\varepsilon; a.a, a.a)}))}$$

The right-hand side appears ill-typed, because $\mathsf{case}(-; a.a, a.a)$ does not send elements of $A + B$ to A. But we may assume that $N_i^\varepsilon\langle r/y\rangle \doteq \mathsf{inl}(M) \in A + B\ [\Psi \mid r_i = \varepsilon]$, in which case N_i^ε evaluates to an $\mathsf{inl}(-)$, and the case to an element of A.

3.5 Related Work

Finally, I intend to compare the cubical meaning explanations to other models of higher-dimensional type theories. As a starting point, can we construct a model of book HoTT by translating every $\Gamma \vdash M : A$ to $\Gamma' \gg M' \in A'\ [\cdot]$? Such a result certainly requires a Kan, univalent universe of Kan types (and a proof of the univalence axiom); a proof of the seg_β law (Section 2.1); and Evan Cavallo's construction[7] of a book-HoTT-style identity type, defined as a higher inductive type family generated by refl. These proofs are excellent test cases for the RedPRL proof assistant [54].

Huber [33] has proven a canonicity result for the formal higher-dimensional type theory of Cohen et al. [16] by establishing typed computability predicates by induction on derivations. While our proofs are structured very differently, Simon Huber and I have noticed similar ideas, most notably the coherent aspects requirement. I intend to closely examine the relationship between our arguments. Modulo the differences in our cube operations and Kan operations, can the cubical meaning explanations (in principle) form a proofs-as-programs interpretation of Cohen et al. [16]?

Our cubical judgmental structure is inspired by the mathematical notion of cubical sets [34], typically presented as presheaves over a cube category. Naïvely one might hope that every A type $[\Psi]$ forms a cubical set $\{M \mid M \approx^\Psi_A M\}_\Psi$, where the functorial action of $\psi : \Psi' \to \Psi$ sends M to V for $M\psi \Downarrow V$. This fails for two reasons. First, the functorial action is only associative up to the relation $\approx^\Psi_A$, not on the nose; we can restore associativity by quotienting each set by its PER. Second, as a natural consequence of dependency, A can contain dimension variables, and therefore can be affected by ψ. Todd Wilson observed[8] that the resulting structure forms a presheaf over the cube category *sliced over* Ψ—it sends every $\psi : \Psi' \to \Psi$ to the set of equivalence classes of $\approx^{\Psi'}_{A_0}$, where $A\psi \Downarrow A_0$. I intend to write up these results and compare these structures to those found in the mathematical literature.

[7]Personal communication. Semantically, the construction relates to a suggestion of Steve Awodey to model book-HoTT-style identity types as the fibrant replacement of the diagonal.

[8]Personal communication.

Bibliography

[1] Stuart F. Allen. *A Non-Type-Theoretic Semantics for Type-Theoretic Language*. PhD thesis, Cornell University, 1987. URL `https://ecommons.cornell.edu/handle/1813/6706`.

[2] Thorsten Altenkirch and Ambrus Kaposi. Towards a cubical type theory without an interval. Preprint, 2015. URL `https://akaposi.github.io/towards_a_cubical_tt_without_interval.pdf`.

[3] Thorsten Altenkirch, Paolo Capriotti, and Nicolai Kraus. Extending Homotopy Type Theory with Strict Equality. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-022-4. doi: http://dx.doi.org/10.4230/LIPIcs.CSL.2016.21. URL `http://drops.dagstuhl.de/opus/volltexte/2016/6561`.

[4] Abhishek Anand, Mark Bickford, Robert L. Constable, and Vincent Rahli. A type theory with partial equivalence relations as types. Presented at *20th International Conference on Types for Proofs and Programs (TYPES 2014)*, 2014. URL `http://www.nuprl.org/documents/Anand/ATypeTheoryWithPartialEquivalenceRelationsAsTypes.pdf`.

[5] Carlo Angiuli and Robert Harper. Meaning explanations at higher dimension. *Indagationes Mathematicae*, November 2016. URL `http://www.cs.cmu.edu/~cangiuli/papers/brouwer.pdf`. Draft, to appear in the special issue *L.E.J. Brouwer, fifty years later*.

[6] Carlo Angiuli and Robert Harper. Computational higher type theory II: Dependent cubical realizability. Preprint, April 2017. URL `http://arxiv.org/abs/1606.09638`.

[7] Carlo Angiuli, Edward Morehouse, Daniel R. Licata, and Robert Harper. Homotopical patch theory. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming*, ICFP '14, pages 243–256, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2873-9. doi: 10.1145/2628136.2628158. URL `http://doi.acm.org/10.1145/2628136.2628158`.

[8] Carlo Angiuli, Robert Harper, and Todd Wilson. Computational higher type

theory I: Abstract cubical realizability. Preprint, June 2016. URL `http://arxiv.org/abs/1604.08873`.

[9] Carlo Angiuli, Edward Morehouse, Daniel R. Licata, and Robert Harper. Homotopical patch theory. *Journal of Functional Programming*, 26, 2016. doi: 10.1017/S0956796816000198.

[10] Carlo Angiuli, Robert Harper, and Todd Wilson. Computational higher-dimensional type theory. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 680–693, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4660-3. doi: 10.1145/3009837.3009861. URL `http://doi.acm.org/10.1145/3009837.3009861`.

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

[12] Henning Basold, Herman Geuvers, and Niels van der Weide. Higher inductive types in programming. *Journal of Universal Computer Science*, 23(1):63–88, January 2017. URL `http://www.jucs.org/jucs_23_1/higher_inductive_types_in`.

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

[14] Guillaume 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.

[15] Guillaume 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`.

[16] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders 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.

[17] Robert L. Constable. *Naïve Computational Type Theory*, pages 213–259. Springer Netherlands, Dordrecht, 2002. ISBN 978-94-010-0413-8. doi: 10.1007/978-94-010-0413-8_7. URL `http://dx.doi.org/10.1007/978-94-010-0413-8_7`.

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

[19] Robert L. Constable, et al. *Implementing Mathematics with the Nuprl Proof*

Development Environment. Prentice-Hall, 1985.

[20] Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Computation*, 76(2):95 – 120, 1988. ISSN 0890-5401. doi: http://dx.doi.org/10.1016/0890-5401(88)90005-3. URL `http://www.sciencedirect.com/science/article/pii/0890540188900053`.

[21] B. A. Davey and H. A. Priestley. *Introduction to lattices and order*. Cambridge University Press, Cambridge, UK, 2002. ISBN 0-521-78451-4. URL `http://opac.inria.fr/record=b1077513`.

[22] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob 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`.

[23] Florian Faissole and Bas Spitters. Synthetic topology in homotopy type theory for probabilistic programming. In *CoqPL 2017*, 2017. URL `http://www.cs.au.dk/~spitters/ProbProg.pdf`.

[24] Nicola Gambino and Richard 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`.

[25] Robert Harper. Constructing type systems over an operational semantics. *J. Symb. Comput.*, 14(1):71–84, July 1992. ISSN 0747-7171. doi: 10.1016/0747-7171(92)90026-Z. URL `http://dx.doi.org/10.1016/0747-7171(92)90026-Z`.

[26] Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, second edition, 2016.

[27] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory*. Oxford University Press, 1998.

[28] Kuen-Bang Hou (Favonia) and Michael Shulman. The Seifert-van Kampen Theorem in Homotopy Type Theory. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-022-4. doi: 10.4230/LIPIcs.CSL.2016.22. URL `http://drops.dagstuhl.de/opus/volltexte/2016/6562`.

[29] Kuen-Bang Hou (Favonia), Eric Finster, Daniel R. Licata, and Peter LeFanu Lumsdaine. A mechanization of the Blakers-Massey connectivity theorem in homotopy type theory. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 565–574, New York, NY, USA,

2016. ACM. ISBN 978-1-4503-4391-6. doi: 10.1145/2933575.2934545. URL `http://doi.acm.org/10.1145/2933575.2934545`.

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

[31] Douglas 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.

[32] Douglas J. Howe and Scott D. Stoller. An operational approach to combining classical set theory and functional programming languages. In and J. C. Mitchell 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.

[33] Simon Huber. *Cubical Interpretations of Type Theory*. PhD thesis, University of Gothenburg, November 2016. URL `http://hdl.handle.net/2077/48890`.

[34] Daniel 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`.

[35] Chris Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of univalent foundations (after Voevodsky). Preprint, June 2016. URL `https://arxiv.org/abs/1211.2851`.

[36] Dan Licata. Weak univalence with "beta" implies full univalence, September 2016. URL `https://groups.google.com/d/msg/homotopytypetheory/j2KBIvDw53s/YTDK4D0NFQAJ`. Email to Homotopy Type Theory mailing list.

[37] Daniel R. Licata. What is homotopy type theory?, 2014. URL `http://dlicata.web.wesleyan.edu/pubs/l14coq/l14coq.pdf`. Talk at Coq Workshop.

[38] Daniel R. Licata and Guillaume 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.

[39] Daniel R. Licata and Robert 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`.

[40] Daniel R. Licata and Michael Shulman. Calculating the fundamental group of the circle in homotopy type theory. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '13, pages 223–232, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-0-7695-5020-6. doi: 10.1109/LICS.2013.28. URL `http://dx.doi.org/10.1109/LICS.2013.28`.

[41] Peter LeFanu 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`.

[42] Zhaohui Luo. Notes on universes in type theory, October 2012. URL `http://www.cs.rhul.ac.uk/~zhaohui/universes.pdf`. Notes for a talk at the Institute for Advanced Study.

[43] Per Martin-Löf. An intuitionistic theory of types: predicative part. In H.E. Rose and J.C. 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.

[44] Per Martin-Löf. Constructive mathematics and computer programming. In L. Jonathan Cohen, Jerzy Łoś, Helmut Pfeiffer, and Klaus-Peter 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`.

[45] Per 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.

[46] Bengt Nordström, Kent Petersson, and Jan Smith. *Programming in Martin-Löf's Type Theory*. Oxford University Press, 1990.

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

[48] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013. ISBN 9781107017788.

[49] A. M. Pitts. Nominal Presentation of Cubical Sets Models of Type Theory. In H. Herbelin, P. Letouzey, and M. Sozeau, editors, *20th International Conference on Types for Proofs and Programs (TYPES 2014)*, volume 39 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 202–220, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-88-0. doi: http://dx.doi.org/10.4230/LIPIcs.TYPES.2014.202. URL `http://drops.dagstuhl.de/opus/volltexte/2015/5498`.

[50] Mike 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.

[51] Mike Shulman. Elementary $(\infty, 1)$-topoi, April 2017. URL `https://golem.ph.`

utexas.edu/category/2017/04/elementary_1topoi.html. Blog post on the n-category café.

[52] Jan M. Smith. The independence of Peano's fourth axiom from Martin-Löf's type theory without universes. *The Journal of Symbolic Logic*, 53(3):840–845, 1988. ISSN 00224812. URL http://www.jstor.org/stable/2274575.

[53] S.F. Smith. *Partial Objects in Type Theory.* PhD thesis, Cornell University, Ithaca, NY, 1989.

[54] Jonathan Sterling, Danny Gratzer, Vincent Rahli, Darin Morrison, Eugene Akentyev, and Ayberk Tosun. RedPRL – the People's Refinement Logic. http://www.redprl.org/, 2016.

[55] 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.

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

[57] The NuPRL Project. Prl project, 2016. URL http://nuprl.org.

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

[59] Benno van den Berg and Richard 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.

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

[61] Vladimir 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.

[62] Vladimir Voevodsky. A simple type system with two identity types. Lecture notes, February 2013. URL https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/HTS.pdf.