Cubical Computational Semantics for Higher Inductive Types

Evan Cavallo
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
ecavallo@cs.cmu.edu

Robert Harper
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
rwh@cs.cmu.edu

Abstract
Homotopy type theory (HoTT) proposes higher inductive types (HITs) as a means of reasoning about higher-dimensional objects in type theory. As with the univalence axiom, however, HoTT does not specify the computational behavior of HITs. Computational interpretations have now been provided for univalence and specific HITs by way of cubical type theories, which use a judgmental infrastructure of dimension names. We extend Angiuli et al.'s relational semantics for cubical type theory with a general class of cubical inductive types (CITs), which can serve as a semantics for HITs either in HoTT or a cubical type theory. We obtain a canonicity theorem: any zero-dimensional term in a CIT evaluates to a point constructor. We suggest a further extension to indexed inductive types by giving semantics for the homotopy fiber family, which is interdefinable with the identity type family.

2012 ACM Subject Classification Theory of computation → Operational semantics

Keywords and phrases Cubical type theory, higher inductive types, computational type theory

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23


Introduction
Intensional Martin-Löf type theory (ITT) [29, §2] internalizes equality with identity types: for \( M, N : A \), the type \( \text{Id}_A(M, N) \) is inhabited when they are in a sense interchangeable. By definition, \( \text{Id}_A(-,-) \) is the least reflexive “relation” on \( A \), generated by a constructor \( a : A \vdash \text{refl}(a) : \text{Id}_A(a,a) \). However, \( \text{Id}_A(-,-) \) is not a mere relation: the Hofmann-Streicher groupoid model [20] and the homotopical models of Awodey-Warren [6] and Voevodsky [25] are interpretations of ITT where \( \text{Id}_A(M,N) \) can have distinct elements. In this case, elements are means of identifying \( M \) and \( N \). The homotopical models interpret types as spaces,
with iterated identity types—e.g., Id_{A(M,N)}(P,Q)—interpreted as higher-dimensional path spaces. Inspired by these models, homotopy type theory (HoTT) [36] extends ITT with the univalence axiom and higher inductive types, making the higher structure visible. The former characterizes the identity types of universes, while the latter allows inductive definitions of spaces with higher structure.

From a computational perspective, HoTT is under-specified: it introduces new elements of identity types without explaining how the identity type eliminator computes with them. For some time, no computational models of HoTT were known. The fog finally began to clear with the Bezem-Coquand-Huber (BCH) model in cubical sets [8], which evaded the constructivity issues of Voevodsky’s model in simplicial sets. This sparked the pursuit of cubical type theories and semantics [10, 4, 30, 2, 11, 1], with the Cohen-Coquand-Huber-Mörtberg (CCHM) type theory [13, 22], Angiuli-Harper-Favonia (AHF) computational semantics [3], and Angiuli et al, type theory [1] providing computational justifications for the univalence axiom based on various notions of cube. In addition to providing computational interpretations of HoTT, cubical theories have practical advantages for synthetic homotopy theory first explored by Brunerie and Licata [27].

Our contribution is to extend the AHF semantics, which currently supports Π and Σ types, path types which internalize dimension abstraction, and univalent universes, to accommodate a general class of higher inductive types. Our schema includes cubical analogues of many HITs which have been proposed for HoTT: the circle, truncations, W-quotients [33], localizations [32], and so on. By nature of the semantics, we obtain a canonicity theorem, which in our case states that any point in a HIT evaluates to a constructor. We suggest an extension to indexed (higher) inductive types by giving semantics for the homotopy fiber family, an inductively generated family interdefinable with the Martin-Löf identity type. This makes it possible to interpret ITT, and thus HoTT with a wide range of HITs, in the AHF semantics.

This paper contains the main ideas; the details are in our preprint [12].

2 Inductively generated spaces and cubical type theory

We will begin by introducing, informally, a representative example of an “inductively-generated space:” the homotopy pushout of a span. A span is a triple of spaces A, B, C and maps \( F : C \to A \) and \( G : C \to B \). Given such data, their (homotopy) pushout \( \Pi = A \amalg_C^E B \) consists of a copy of A, a copy of B, and, for every \( c \in C \), a path connecting \( Fc \in A \) to \( Gc \in B \). Put another way, the copies of A and B in \( \Pi \) are connected by a C-shaped cylinder, which is attached to the copies according to \( F \) and \( G \).

![Figure 1](image_url) The pushout \( A \amalg_C^E G B \), with \( Fc \in A \) and \( Gc \in B \) connected by a line for \( c \in C \).

In cubical type theory and semantics, paths are exposed as a primitive concept. Judgments are parameterized by a context \( \Psi \) of dimension (or interval) names \( x, y, z, \ldots \), each of which is thought of as varying in the unit interval \( [0, 1] \). Dimension terms \( r, s, \ldots \) are either dimension names or the constants 0 or 1. A term \( M \in A[\Psi,x] \) which depends on a name \( x \) is a line in \( A \), and its endpoints are the terms \( M(0/x) \) and \( M(1/x) \) obtained by substituting a constant for \( x \). In cubical type theory, we can describe the pushout as inductively generated by three
The pushout is only one of a diverse array of higher inductive constructions: there is the torus, which has a 2-dimensional “square” constructor, or the truncations which have recursive path constructors. This paper is concerned with the computational semantics of such types, which we will call cubical inductive types (CITs) in the cubical setting. We work in the AFH framework, which we review in Section 3. In Section 4, we define a schema which encompasses the pushout as well as types such as n-truncations which require recursive constructors. We define the denotation of each instance of the schema, define the operational semantics of hcom, coe, and elim, and prove their type correctness. In doing so, we obtain a
canonicity theorem: any zero-dimensional term in a CIT evaluates to a point constructor. In Section 5, we define the homotopy fiber type, which is interdefinable with the identity type. Our definition is inspired by considering the fiber type as a particular indexed inductive type. Adding the homotopy fiber type makes it possible to interpret HoTT in our semantics. There is no precise definition of HIT in [36], but our schema suffices to interpret every ordinary (parameterized, but not indexed or inductive-inductive) HIT used therein.

3 Computational higher type theory

The AFH framework, dubbed computational higher type theory (CHiTT), provides a relational semantics of cubical type theories, extending the Nuprl approach to semantics with dimension names. In this approach, a type denotes a collection of values in an untyped programming language. In CHiTT, the untyped language is a cubical language with dimension names, and the denotation of a type is indexed by dimension contexts and equipped with Kan structure.

3.1 Cubical programming language

A cubical programming language consists of terms $M$ and a deterministic operational semantics on closed terms, given by judgments $M \rightarrow M'$ ($M$ steps to $M'$) and $M \text{ val} (M$ is a value). These terms can contain dimension names: for example, $\text{glue}_V(P) \text{ val}$. We assume equality of names is decidable. We write $M \text{ tm } [\Psi]$ to mean that $FD(M) \subseteq \Psi$, $FD(M)$ being the set of names free in $M$. We use $\rightarrow^*$ for the transitive closure of $\rightarrow$, and write $M \Downarrow V$ to mean that $M \rightarrow^* V$ with $V$ val.

We write $\psi : \Psi \rightarrow \Psi$ for dimension substitutions taking $M \text{ tm } [\Psi]$ to $M\psi \text{ tm } [\Psi']$. We call these $M\psi$ aspects of $M$. The operational semantics need not be stable under substitution: $M \rightarrow M'$ does not imply $M\psi \rightarrow M'\psi$, nor does $M \text{ val}$ imply $M\psi \text{ val}$. For example, $\text{glue}_V(P)$ will be a value, while $\text{glue}_V(P)$ will step to $\text{left}_V(FP)$.

We will start with the cubical language used in [3] and extend it with new operators as needed. The theorems we prove about it will hold for any language which contains the fragment we present. One could instead use Church-style encodings, but some extension to the $\lambda$-calculus is needed to accommodate names. First, most obviously, one needs dimension abstraction $\langle x \rangle M$ and application $M \langle x \rangle r$ with $\langle x \rangle M \langle x \rangle r \rightarrow M[r/x]$. In fact, a second abstractor $\langle x \rangle M$ is needed which evaluates under its binder: $\langle x \rangle M \rightarrow \langle x \rangle M'$ when $M \rightarrow M'$. This is necessary for $\text{coe}$, which evaluates its type argument. Finally, one would need an extended $\lambda$-abstractor $\lambda^{r = r'}$ which reduces when an equation holds.

\[
\begin{align*}
\lambda^{r = r'}(a.M; N) \text{ val} & \quad r \neq r' & \quad \lambda^{r = r'}(a.M; N)(P) \rightarrow M[P/a] & \quad r = r' \\
\end{align*}
\]

For our purposes, there is no need for frugality, so we will simply introduce operators and operational semantics as needed.

3.2 Cubical type systems

Given a programming language, we can define a type system over it. In Nuprl-style semantics, the denotation of a type $A$ is a partial equivalence relation (PER) $[A]$ on values. We say that $V$ is a canonical value in $A$ if $[A](V, V)$, and that $V, V'$ are equal canonical values if $[A](V, V')$; the use of PERs is a convenience to simultaneously carve out a set of values and impose an equivalence relation upon it. A term $M$ is in $A$ if it evaluates to a canonical value in $A$, and $M, M'$ are equal in $A$ if they evaluate to equal such values.
In the cubical setting, the elements of a type \( A \) may contain dimension names, so we stratify its denotation as a family of PERs \([[A]_\Psi] \) indexed by dimension context \( \Psi \). At each \( \Psi \), \([[A]_\Psi] \) relates values \( V \) with \( \text{FD}(V) \subseteq \Psi \), specifying the equal canonical values in that context.

This is the special case where \( A \) itself is free of dimension names: in general, the denotation of a type \( A \) in context \( \Psi \) is a family of PERs \([[A]_\Psi] \) indexed by substitutions \( \psi : \Psi' \rightarrow \Psi \). For each \( \psi : \Psi' \rightarrow \Psi \), \([[A]_\Psi] \) specifies the values of the aspect \( A_{\psi} \) in context \( \Psi' \). We call such a family a \( \Psi \)-relation (in this case, a value \( \Psi \)-PER). We write \( \alpha, \beta, \ldots \) for \( \Psi \)-relations.

We abbreviate \( \alpha_{\psi}(M, M') \) as \( \alpha_{\psi}(M) \) and \( \alpha_{\id}(M, M') \) as \( \alpha(M, M') \). If \( \alpha \) is a \( \Psi \)-relation and \( \psi : \Psi' \rightarrow \Psi \), we write \( \alpha_{\psi} \) for the \( \Psi' \)-relation \( (\alpha_{\psi})_{\psi'} := \alpha_{\psi_{\psi'}} \).

As in the zero-dimensional case, a value \( \Psi \)-PER \( \text{Trm}(\alpha) \) on terms.

Here, however, we don’t want to include every term which evaluates to a value in \( \alpha \). For example, suppose that \( V \) is some value in \( \alpha \). Using the \( \lambda^{r=x} \) operator defined above, we can write the term \( M := (\lambda^{x=0}(V; \lambda_{\subseteq 5}))() \), for which we have \( M \Downarrow V \) but \( M(0/x) \Downarrow 5 \). We want to exclude such terms from \( \text{Trm}(\alpha) \), only including terms all of whose aspects evaluate into \( \alpha \). Moreover, they should do so in a coherent way, giving the same result up to \( \alpha \) no matter how one interleaves substitution and evaluation.

**Definition 1.** Let \( \alpha \) be a value \( \Psi \)-relation. We define \( \text{Trm}(\alpha_{\psi})(M, M') \) to hold for \( \psi : \Psi' \rightarrow \Psi \) when for every \( \psi_1 : \Psi_1 \rightarrow \Psi' \) and \( \psi_2 : \Psi_2 \rightarrow \Psi_1 \), we have \( M_{\psi_1} \Downarrow V_1, V_1 \Downarrow V_2, M_{\psi_1} \Downarrow V_2, M'_{\psi_1} \Downarrow V_2', M'_{\psi_1} \Downarrow V_2', M'_{\psi_1} \Downarrow V_2' \), and \( M'_{\psi_1} \Downarrow V_2' \) for some \( V_1, V_2, V_2', V_2' \), such that \( \psi_{\psi_1} \psi_{\psi_2}(W, W') \) for every \( W, W' \in \{V_2, V_2', V_2', V_2'\} \).

We will not work with this definition directly. Instead, we will present an interface of lemmas, proven in Appendix A of our preprint, which should help to give an intuition. First, note that if \( \text{Trm}(\alpha_{\psi})(M, M') \), then in particular \( M, M' \) evaluate to values related by \( \alpha \). The \( \Psi \)-relation \( \text{Trm}(\alpha) \) is always stable, where a \( \Psi \)-relation \( \beta \) is stable when \( \beta_{\psi}(M, M') \) implies \( \beta_{\psi_{\psi'}}(M'_{\psi}, M'_{\psi'}) \) for all \( \psi, \psi', M, M' \). The value \( \Psi \)-relations we use will typically not be stable, as valuehood itself is not stable: \( \text{glue}^0_{\Psi_1}(P) \) may be in \([\Psi_2]\), but \( \text{glue}^0_{\Psi_1}(P) \) is not even a value.

**Lemma 2 (Introduction).** Let \( \alpha \) be a value \( \Psi \)-PER. If for every \( \psi : \Psi' \rightarrow \Psi \), either \( \alpha_{\psi}(M, M') \) or \( \text{Trm}(\alpha)_{\psi}(M, M') \), then \( \text{Trm}(\alpha)(M, M') \).

In particular, if \( \alpha_{\psi}(M, M', \psi) \) for all \( \psi \), then \( \text{Trm}(\alpha)(M, M') \). Thus, for example, we have \( \text{Trm}(\Pi)(\text{left}_{\Pi}(M), \text{left}_{\Pi}(M')) \) when \( \text{Trm}(\Pi)(M, M') \). We say \( \alpha \) is value-coherent if it satisfies the stronger condition that \( \alpha \subseteq \text{Trm}(\alpha) \). We will require this property of all types; it will fail, for example, if \( \alpha \) contains \( \text{glue}^0_{\Pi}(P) \) but not \( \text{left}_{\Pi}(FP) \).

**Lemma 3 (Coherent expansion).** Let \( \alpha \) be a value \( \Psi \)-PER and let \( M, M' \) \( \text{tm} \ \Psi \). If for every \( \psi : \Psi' \rightarrow \Psi \), there exists \( M'' \) such that \( M_{\psi} 
Rightarrow M'' \) and \( \text{Trm}(\alpha)_{\psi}(M'', M') \), then \( \text{Trm}(\alpha)(M, M') \).

In particular, if \( M_{\psi} \nRightarrow M' \) for all \( \psi \) and \( \text{Trm}(\alpha)(M, M') \), then \( \text{Trm}(\alpha)(M, M') \). Thus \( \text{Trm}(\Pi)(\text{glue}^0_{\Pi}(P), \text{left}_{\Pi}(FP)) \) when \( \text{Trm}(\Pi)(\Pi)(P, P') \). Using this and the corresponding \( (1/x) \) reduction, we can show \( \text{Trm}(\Pi)(\text{glue}^0_{\Pi}(P), \text{glue}^0_{\Pi}(P')) \) for \( \text{Trm}(\Pi)(P, P') \) and \( \varepsilon \in \{0, 1\} \).

Using Lemma 2, we can then prove \( \text{Trm}(\Pi)(\text{glue}^0_{\Pi}(P), \text{glue}^0_{\Pi}(P')) \) for any \( \text{Trm}(\Pi)(P, P') \).

Note that \( \text{Trm}(\alpha) \) is not closed under arbitrary expansion, so we cannot show that \( (\lambda^{x=0}(\varepsilon_{\text{glue}^0_{\Pi}(P); \lambda_{\subseteq 5}))()) \) is in \( \text{Trm}(\Pi) \).

A candidate cubical type system carves out a universe of \( \Psi \)-PERs and gives them syntactic names. Precisely, a candidate is a family \( \tau = (\tau_{\psi})_{\Psi} \) of three-place relations \( \tau_{\psi}(A_0, B_0, \psi) \) relating values \( A_0, B_0 \) \( \text{tm} \ \Psi \) and PERs \( \varphi \) on values \( V, V' \) in context \( \Psi \). As with \( \text{Trm} \) in the case of value \( \Psi \)-PERs, a candidate induces relations \( \text{PTy}(\tau)(A, B, \alpha) \) on terms \( A, B \) \( \text{tm} \ \Psi \).

---

**CVIT 2016**
and value $\Psi$-PERs $\alpha$. We will again omit the definition, but note that $PTy(\tau)_{\psi}(A, B, \alpha)$ implies $A \downarrow A_0$ and $B \downarrow B_0$ with $\tau_{\psi}(A_0, B_0, \alpha_{id_{\psi}})$ as well as $PTy(\tau)_{\psi}(A_{\bar{\psi}}, B_{\bar{\psi}}, \alpha_{\bar{\psi}})$ for any $\psi : \Psi \rightarrow \Psi$. A candidate $\tau$ is a cubical type system (CTS) when a given $A_0, B_0$ are related by $\tau_{\psi}$ to at most one $\psi$, each $\tau_{\psi}(-, -, \phi)$ is a PER, and $PTy(\tau)_{\psi}(A, B, \alpha)$ implies that $\alpha$ is value-coherent. We then write $[A]$ for the unique $\alpha$ with $PTy(\tau)_{\psi}(A, A, \alpha)$ when it exists.

The AHF semantics is a cumulative hierarchy of cubical type systems, each of which appears as a universe type in the next.

We now define standard type-theoretic judgments relative to a CTS $\tau$. First, we say $A$ and $A'$ are equal pretypes and write $A \equiv A'$ type pre $[\Psi]$ when $PTy(\tau)_{\psi}(A, A', \alpha)$ for some $\alpha$. We write $A$ type pre $[\Psi]$ to mean $A \equiv A$ type pre $[\Psi]$. Presupposing $A$ type pre $[\Psi]$, we say $M \equiv M' \in A[\Psi]$ when $TM(\{A\})(M, M')$, writing $M \in A[\Psi]$ for $M \equiv M \in A[\Psi]$. We define the open judgments by functionality: open terms are equal when they send equal closing substitutions to equal results. As in Kripke semantics, we must also quantify over dimension substitutions. For example, we define $a : A \Rightarrow B \Rightarrow B'$ type pre $[\Psi]$ to hold when $B_{\psi}(M[A] \Rightarrow B'_{\psi}[M'[A]]$ type pre $[\Psi]$ holds for every $\psi : \Psi' \rightarrow \Psi$ and $M \equiv M' \in A_{\psi}[\Psi']$. We leave it to the reader to infer the definitions of context equality $\Gamma \equiv \Gamma'$ ctx pre $[\Psi]$, equality in contexts $\bar{M} \equiv M \in \Gamma[\Psi]$, and the general open judgments $\Gamma \Rightarrow A \Rightarrow A'$ type pre $[\Psi]$ and $\Gamma \Rightarrow N \equiv N' \in A[\Psi]$. We will use the notation $\gamma : \Gamma$ to refer to the variables in $\Gamma$ as a group.

Finally, we can use this notation to concisely state an elimination lemma for $TM$.

**Definition 4.** We say that $\alpha \vdash N \in TM[\Psi]$ (a term $N$ with one free variable $a$) is *eager* if for all $\psi : \Psi' \rightarrow \Psi$ and $M \vdash TM[\Psi']$, we have $N\psi[M/a] \downarrow W$ iff there exists $V \in TM[\Psi]$ such that $M \downarrow V$ and $N\psi[V/a] \downarrow W$.

**Lemma 5 (Elimination).** Let $A$ type pre $[\Psi]$ and $a : A \Rightarrow B$ type pre $[\Psi]$, and let $a \vdash N, N' \in TM[\Psi]$, then $N \psi[V/a] \equiv N' \psi[V'/a] \in B_{\psi}(V, V')$. Then $a : A \Rightarrow N \equiv N' \in B[\Psi]$.

### 3.3 Kan types

We now have a cubical language and pretypes which define collections of cubes. Finally, we distinguish (Kan) types as those pretypes which behave support Kan operations.

First, we have the homogeneous composition $hcom$, which adjusts a term along a tube. A tube is a list $(\xi_1 \rightarrow y, N_1, \ldots, \xi_n \rightarrow y, N_n)$ of constraints $\xi_i$ of the form $r_i = r_j$ paired with lines (terms in an abstracted $y$). We write $\vdash \xi$ to mean that a constraint holds, use $\Xi$ for lists of constraints and write $\vdash \Xi$ when $\vdash \xi$ for all $\xi \in \Xi$. We define restricted judgments $A \equiv A'$ type pre $[\Psi] \equiv \Xi$ and $M \equiv M' \in A[\Psi] \equiv \Xi$ to hold $A \equiv A'$ type pre $[\Psi]$ and $M \psi = M' \psi \in A_{\psi}[\Psi']$ respectively hold for all $\psi : \Psi' \rightarrow \Psi$ with $\vdash \Xi\psi$. We similarly write $\alpha_{\psi}[\Xi](M, M')$ to mean that $\alpha_{\psi}(M, M')$ holds for all $\psi : \Psi' \rightarrow \Psi$ with $\vdash \Xi\psi$. A tube $\xi_i \rightarrow y, N_i$ well-typed in $A$ at $\Psi$ when $N_i \equiv N_j \in A[\Psi, y | \xi_i, \xi_j]$ for each $i, j$—in particular, $N_i \in A[\Psi, y | \xi_i]$ for all $i$. These equations ensure that the terms $N_i$ agree where the constraints overlap, so that a tube is a “partial line” defined on their union. A constraint list $\Xi$ is valid when there exists some $r$ such that either $(r = r) \in \xi_i$ or both $(r = 0) \in \Xi$ and $(r = 1) \in \Xi$. We require tubes to have valid constraint sets, for reasons discussed in Section 4.5.

We say $M \in A[\Psi]$ is a cap for $\xi_i \rightarrow y, N_i$ at $r$ when $M \equiv N_i(r/y) \in A[\Psi, y | \xi_i]$ for each $i$. Intuitively, this means $M$ fits into the tube at position $r$. The $hcom$-Kan condition requires that, given a valid tube and a cap at $r$, we can produce a cap at $r'$. Syntactically, we ask that $hcom{\bar{\psi}}^r(M; \xi_i \rightarrow y, N_i) \in A[\Psi]$ with $hcom{\bar{\psi}}^r(M; \xi_i \rightarrow y, N_i) \equiv N_i(r'/y) \in A[\Psi, y | \xi_i]$ for all $i$. We require that degenerate $hcom$s trivialize: $hcom{\bar{\psi}}^r(M; \xi_i \rightarrow y, N_i) \equiv M \in A[\Psi]$.
Second, the **coe-Kan condition** requires that for every \( r, r' \), and \( M \in A(r/x) [\Psi] \), we have \( \text{coe}^r_{r'} (M) \in A(r'/x) [\Psi] \). As with hcom, a degenerate coe must be trivial: 
\[
\text{coe}^r_r (M) = M \in A(r/x) [\Psi].
\]

Equal pretypes \( A \doteq A' \text{ type}_{\text{pre}} [\Psi] \) are **equally Kan**, written \( A \doteq A' \text{ type}_{\text{kan}} [\Psi] \) when their aspects satisfy the Kan conditions with equal implementations of hcom and coe. We write \( \Gamma \doteq \Gamma' \text{ ctkan} [\Psi] \) to mean that each pair in \( \Gamma, \Gamma' \) is equally Kan.

## 4 Cubical inductive types in generality

### 4.1 The schema

To motivate the definition of our schema, let us first consider the zero-dimensional case. Existing schemata for inductive types specify an inductive type with a list of constructors, with recursive constructor arguments restricted to a grammar of *strictly positive* types to ensure consistency [17, 18]. A constructor for a type \( X \) could thus be specified by a pair \( (\Gamma, \gamma, \Theta) \), where \( \gamma : \Gamma = (x_1 : A_1, \ldots, x_n : A_n) \) is a telescope of non-recursive argument types and \( \Theta = (\lambda_1, \ldots, \lambda_k) \) lists the recursive argument types per a grammar \( \Lambda ::= X | (a:A) \rightarrow A \).

For cubical inductive types, we need to make a number of extensions. First, we must allow for dimension parameters to constructors. Second, we need to be able to specify a set of boundary constraints on these parameters and terms to which the boundary is attached. As such, a constructor will take the form of a triple \( (\Gamma; \gamma, \Theta; \xi, \xi_k \leftrightarrow \gamma, \Theta, M_k) \), with \( \xi_i \) the dimension parameters, \( \xi_k \) the boundary constraints, and \( M_k \) the corresponding terms. The grammar of argument types becomes a formal type theory of *argument types* \( \Lambda \) inhabited by *argument terms* \( M \). Unlike the zero-dimensional case, where constructors are independent of each other, here we must allow boundary terms to refer to previously given constructors.

We define the schema relative to a fixed cubical type system \( \tau \). The central judgment \( \mathcal{K} \equiv \mathcal{K}' \text{ constrs} [\Psi] \), defined below, states that \( \mathcal{K} \) and \( \mathcal{K}' \) are equal lists of labelled constructors.

\[
\frac{\mathcal{K} \equiv \mathcal{K}' \text{ constrs} [\Psi]}{[\mathcal{K}, \ell : C] \equiv [\mathcal{K}', \ell : C'] \text{ constrs} [\Psi]}
\]

This judgment is mutually inductively defined with a judgment \( \mathcal{K} \vdash C \equiv C' \text{ constr} [\Psi] \) asserting that \( C \) and \( C' \) are equal constructors over a prefix \( \mathcal{K} \). We draw labels \( \ell \) from a fixed set \( L \), writing \( \ell \in \mathcal{K} \) to mean that \( \ell \) occurs in \( \mathcal{K} \) and \( \mathcal{K}[\ell] \) for the constructor labelled \( \ell \).

The constructor equality judgment is itself mutually defined with judgments \( \Lambda \equiv A' \text{ atype} [\Psi] \), \( \theta : \Theta \quad \text{actx} [\Psi] \), and \( \Gamma; \Theta \vdash M \equiv M' : A [\Psi] \).

**Definition 6.** Presupposing \( \mathcal{K} \text{ constrs} [\Psi] \), we say \( \mathcal{K} \vdash C \equiv C' \text{ constr} [\Psi] \) when \( C = (\Gamma; \gamma, \Theta; \xi, \xi_k \leftrightarrow \gamma, \Theta, M_k) \) and \( C' = (\Gamma'; \gamma, \Theta'; \xi, \xi_k \leftrightarrow \gamma, \Theta', M'_k) \) where

1. \( \Gamma \doteq \Gamma' \text{ ctkan} [\Psi] \),
2. \( \gamma : \Gamma \gg \Theta \equiv \Theta' \text{ actx} [\Psi] \),
3. \( \mathcal{F} (\xi_k) \subseteq \xi_i \) and \( \xi_k \) is valid if \(|\xi_i| > 0\),
4. \( \gamma : \Gamma \gg \mathcal{K}; \theta : \Theta \vdash M_k \equiv M'_k : X [\Psi, \xi_i | \xi_k, \xi_l] \) for each \( k, l \).

The judgment \( A \equiv A' \text{ atype} [\Psi] \) is defined below. The argument type \( X \) is the recursive reference to the inductive type.

\[
\frac{X \equiv X \text{ atype} [\Psi]}{A \equiv A' \text{ type}_{\text{kan}} [\Psi] \quad a : A \gg B \equiv B' \text{ atype} [\Psi]}
\]
Constructors and Composition

\[
\mathcal{K}[\ell] = (\Gamma; \gamma. \Phi; \overrightarrow{x_i}, \xi_k \mapsto \gamma. \varphi. M_k) \quad \overrightarrow{P_n} \in \Gamma [\Psi] \quad \mathcal{K}; \Theta \vdash N_j : \Phi [\overrightarrow{P_n} / \gamma] [\Psi]
\]

\[
\mathcal{K}; \Theta \vdash \text{intro}_r (\overrightarrow{P_n}; N_j) : X [\Psi]
\]

\[
\mathcal{K}; \Theta \vdash \text{intro}_r (\overrightarrow{P_n}; N_j) \equiv M_k (\overrightarrow{P_i}/\overrightarrow{X_i}) [\overrightarrow{P_n}/\gamma][\overrightarrow{N_j}/\psi] : X [\Psi | \xi_k]
\]

\[
\mathcal{K}; \Theta \vdash M : X [\Psi]
\]

\[
(\forall i, j) \mathcal{K}; \Theta \vdash N_i \equiv N_j : X [\Psi | \xi_i, \xi_j] \quad (\forall i) \mathcal{K}; \Theta \vdash N_i (r/y) \equiv M : X [\Psi | \xi_i]
\]

Functions

\[
A \text{ type}_{\text{kan}} [\Psi] \quad a : A \Rightarrow \mathcal{K}; \Theta \vdash N : B [\Psi]
\]

\[
\mathcal{K}; \Theta \vdash \lambda a. N : (a:A) \rightarrow B [\Psi]
\]

\[
K; \Theta \vdash \text{app}(N, M) : B[M/a] [\Psi]
\]

\[
K; \Theta \vdash M : (a:A) \rightarrow B [\Psi]
\]

\[
K; \Theta \vdash \text{app}(\lambda a. N, M) : (a:A) \rightarrow B [\Psi]
\]

\[
K; \Theta \vdash M \equiv \lambda a. (\text{app}(M, a)) : (a:A) \rightarrow B [\Psi]
\]

Figure 2 Recursive argument term typing rules. We have omitted variable, structural, and compatibility rules.

Note that this judgment is inductively defined, unlike \( A \Rightarrow A' \text{ type}_{\text{kan}} [\Psi] \), which is defined in terms of coherent evaluation. We use \( \Rightarrow, \equiv, \in \) rather than \( \rightarrow, \equiv, \in \) to emphasize this point, and we use blue text to distinguish the schema language. However, the argument type judgment does use the ordinary type judgment in its definition, and the open form \( \Gamma \Rightarrow A \equiv A' \text{ atype} [\Psi] \) is defined by functionality: we say that \( a : A \Rightarrow B \equiv B' \text{ atype} [\Psi] \) when \( b_\psi[M/a] \equiv b'_\psi[M/a] \text{ atype} [\Psi] \) for every \( \psi : \Psi' \rightarrow \Psi \) and \( M \equiv M' \in A_\psi [\Psi'] \). Where this open form occurs in the definition of \( A \equiv A' \text{ atype} [\Psi] \), one can imagine it replaced by its definition. An argument context \( \Theta \) is a list of argument types; the open form \( \Gamma \Rightarrow \Theta \equiv \Theta' \text{ actx} [\Psi] \) is similarly defined.

We define the well-typed argument terms by a judgment \( \mathcal{K}; \Theta \vdash M \equiv M' : A [\Psi] \) parameterized by a list \( K \) of previous constructors and an argument context \( \Theta \). This judgment is defined in Figure 2. Again, this is an inductive definition, while the “G-open” form \( \Gamma \Rightarrow K; \Theta \vdash M \equiv M' : A [\Psi] \) is defined by functionality. We have access to two kinds of argument terms in the indeterminant type: intro terms, representing constructors defined in \( K \), and fcoms, representing composites in the inductive type. (We will return to fcom in the next section.) The term \( \text{intro}_r (\overrightarrow{P_n}, \overrightarrow{N_j}) \) takes a label \( \ell \) pointing to its definition in \( K \), dimension parameters \( \overrightarrow{P_i} \), non-recursive parameters \( \overrightarrow{P_n} \), and recursive arguments \( \overrightarrow{N_j} \). The function type \( (a:A) \rightarrow B \) is inhabited by \( \lambda \)-terms and supports elimination via application. The \( \beta, \eta \), and boundary equations ascribed to these terms will be validated by their operational equivalents, which we describe in the following section.

This completes the schema definition; we can now try defining some examples. We can specify the pushout \( A \pi^G \ll B \) by 0-constructors left : \( (A; a. \phi) \) and right : \( (B; b. \phi) \).
and a 1-constructor
\[ \text{glue} : (C; c.\emptyset; x) \xrightarrow{\text{intro}\_\text{left}} (Fc; \emptyset), \]
\[ x = 1 \xrightarrow{\text{intro}\_\text{right}} (Gc; \emptyset) . \]

(We use \( \emptyset \) to denote empty lists.) An example with a recursive construction is the \((-1)\)-truncation \( \| A \| \) of a type \( A \), which trivializes the homotopical structure of \( A \). We can specify \( \| A \| \) as having a 0-constructor \( \text{pt} : (A; a.\emptyset; \emptyset) \), which includes \( A \) into \( \| A \| \), and a 1-constructor
\[ \text{path} : (\emptyset; \emptyset; (X, X); x) x = 0 \xrightarrow{\emptyset} (l_0, l_1), t_0 \]
\[ x = 1 \xrightarrow{\emptyset} (l_0, l_1), t_1 , \]
which draws a line \( \text{path}^r(N; N') \) between any pair of elements \( N, N' \in \| A \| \). The type \( \| A \| \)
is the universal mere proposition with a map from \( A \), where a mere proposition is a type in which any pair of elements are connected by a path. Recursive constructors are quite useful for universal constructions of this kind: the truncation is the lowest of a tower of \( n \)-truncations for \( n \geq -1 \), which we can define similarly, and these are themselves instances of a general localization construction we can also encode [32].

4.2 The \( \Psi \)-PER \( i(K) \)
We can now specify CITs; the next step will be to construct them. For each \( K \) constrs \( [\Psi] \), we will now define a value \( \Psi \)-PER \( i(K) \) generated by \( K \). We continue to work relative to a cubical type system \( \tau \); later, we will discuss how to construct a type system which contains \( i(K) \). We build \( i(K) \) from two kinds of values, fcoms and intros, which have operational semantics defined in Figure 3.

\[
\begin{align*}
\text{fcom}^{\tau \rightarrow \tau'} (M; \xi_i) & \iff y.N_i & (\forall i) \not\models \xi_i, r \neq r' \quad & \text{fcom}^{\tau \rightarrow \tau'} (M; \xi_i) \rightarrow y.N_i \rightarrow M \\
\models \xi_i & (\forall j < i) \not\models \xi_j & C = (\Gamma; \gamma.\Theta; \xi_k \rightleftarrows \gamma.\beta.M_k) & (\forall k) \not\models \xi_k (\tau_i / \tau_i) \\
\text{fcom}^{\tau \rightarrow \tau'} (M; \xi_i) \rightarrow y.N_i \rightarrow N_i (r'/y) & \vDash C_{K,\ell} (P_{\ell}; N_j) & \text{intro}^{\text{fcom}}_{\text{c}} (\xi_k (\tau_i / \tau_i)) \rightarrow (\theta.\beta.M_k (\tau_i / \tau_i) | P_{\ell} / / \gamma) K (N_j) \\
C = (\Gamma; \gamma.\Theta; \xi_k \rightleftarrows \gamma.\beta.M_k) & \not\models \xi_k (\tau_i / \tau_i) & (\forall k < l) \not\models \xi_l (\tau_i / \tau_i) \\
\text{intro}^{\text{fcom}}_{\text{c}} (\xi_k (\tau_i / \tau_i)) \rightarrow (\theta.\beta.M_k (\tau_i / \tau_i) | P_{\ell} / / \gamma) K (N_j) \\
\end{align*}
\]

Figure 3 Operational semantics of fcom and intro.

An fcom is a free or formal (homogeneous) composite. Recall that populating a CIT with constructors is not enough to obtain a Kan type: we also need to account for composites. In a Kan type \( A \), a composite \( \text{hcom}^{\tau \rightarrow \tau'} (M; \xi_i \rightarrow y.N_i) \) is required to satisfy two boundary conditions: it must be equal to \( N_i (r'/y) \) when the corresponding \( \xi_i \) holds, and it must be equal to \( M \) when \( r = r' \). The fcom operator satisfies these constraints in the most simple-minded way: \( \text{fcom}^{\tau \rightarrow \tau'} (M; \xi_i \rightarrow y.N_i) \) steps to \( N_i (r'/y) \) when \( \models \xi_i \) for some least \( i \), and to \( M \) when \( r = r' \) (and \( \not\models \xi_i \) for all \( i \)). Otherwise, it is a value. We can make any pretype \( A \) hcom-Kan by adding fcom values to its \( \Psi \)-PER and setting \( \text{hcom}^{\tau \rightarrow \tau'} (M; \xi_i \rightarrow y.N_i) \rightarrow \text{fcom}^{\tau \rightarrow \tau'} (M; \xi_i \rightarrow y.N_i) \). This is the approach we take for CITs.
An intro term takes the form \( \text{intro}_K^\epsilon(P_n; N_j) \), the \( K \) annotation being necessary for the operational semantics. To define the boundary behavior of an intro term, we use the interpretation \( \langle \theta, M \rangle^K(N_j) \) of an argument term \( M \) in free argument variables \( \theta \), which takes \( K \) and instantiations \( N_j \) for the argument variables. This meta-function simply replaces each schematic operator (e.g., \( \text{intro}_\epsilon \)) with the corresponding term \( \langle \text{intro}_K^\epsilon \rangle \) and any \( \theta_j \) with the corresponding \( N_j \). When a boundary constraint for an intro term is satisfied, it steps to the instantiation of the corresponding boundary term. Otherwise, it is a value.

We want \( i(K) \) to be the least \( \Psi \)-relation closed under fcom and intro values. We thus define corresponding operators FCOM and INTRO\(_{K,\epsilon} \) and take \( i(K) \) to be their least-fixed-point.

> **Definition 7.** For any \( \Psi \)-relation \( \alpha \), we define FCOM\( (\alpha)_{\epsilon}(V, V') \) to hold when \( V = \text{fcom}^{\gamma\rightarrow^\epsilon}(M; \xi_i \rightarrow y. N_i) \) and \( V' = \text{fcom}^{\gamma\rightarrow^\epsilon}(M'; \xi_i \rightarrow y. N'_i) \) where

1. \( \tilde{\xi}_i \) is valid, \( \not| \xi_i \) for all \( i \), and \( r \neq r' \),
2. \( \xi_i \rightarrow y. N_i \) and \( \xi_i \rightarrow y. N'_i \) are equal tubes in \( \text{TM}(\alpha\psi) \) with caps \( \text{TM}(\alpha\psi)(M, M') \) at \( r \).

To define INTRO\(_{K,\epsilon} \), we need to instantiate argument types. Given \( A \), we define a term \( \{ A \}(\alpha) \) by \( \{ X \}(\alpha) := A \) and \( \{ (b : B) \rightarrow C \}(\alpha) := \{ c \}(\alpha) \). The \( \Psi \)-relation interpretation \( \{ A \}(\alpha) \) of \( A \) at a \( \Psi \)-relation \( \alpha \) is defined similarly.

> **Definition 8.** Let \( \ell \in K \). For any \( \Psi \)-relation \( \alpha \), we define INTRO\(_{K,\epsilon}(\alpha)_{\epsilon}(V, V') \) to hold when \( V = \text{intro}_K^\epsilon(P_n; N_j) \) and \( V' = \text{intro}_K^\epsilon(P'_n; N'_j) \) where

1. \( K\psi \equiv K' \equiv K''(" \psi \) constrs \( \Psi \) \) and \( K\psi[\ell] = (\Gamma; \gamma, \Theta; \gamma_i, \xi_k \rightarrow \gamma. \theta_i.M_k) \) with \( \Theta = p_j : B_j \),
2. \( \not| \xi_k \rightarrow \gamma_i. \xi_k \) for all \( k \),
3. \( \Gamma \vdash P_n \equiv P'_n \rightarrow \Gamma[\psi] \),
4. \( \{ B_j [P_n/\gamma] \}(\alpha\psi)(N_j, N'_j) \) for all \( j \).

We define \( F_K(\alpha) := \text{FCOM}(\alpha) \cup \bigcup_{\ell \in K} \text{INTRO}_K(\alpha) \) and \( i(K) \) to be the least-fixed-point of \( F_K \).

To include \( i(K) \) in a cubical type system, we must know that it is value-coherent, i.e., that \( i(K) \subseteq \text{TM}(i(K)) \). The proof of this, which validates introduction rules for \( \text{TM}(i(K)) \), naturally interleaves with proofs of boundary equations in \( \text{TM}(i(K)) \). To see this, consider the example of fcom. To show that the term \( F := \text{fcom}^{\gamma\rightarrow^\epsilon}(M; \xi_i \rightarrow y. N_i) \) belongs to \( \text{TM}(i(K)) \) when its constituents are well-typed, we first show that \( \text{TM}(i(K))|_{\psi{i}}(F, N_i) \) for each \( i \) and \( \text{TM}(i(K))|_{\psi=\epsilon}(F, M) \). These each follow from Lemma 3. For the former, for example, we have for any \( \psi' \) with \( \vdash i \xi_i \psi' \) that \( F\psi' \rightarrow N_j \psi' \), where \( j \leq i \) is least with \( \vdash \xi_i \psi' \). For any such \( j \), we have \( \text{TM}(i(K))|_{\psi=\epsilon}(N_j \psi', N_j \psi') \) by typing assumption; thus Lemma 3 gives \( \text{TM}(i(K))|_{\psi{i}}(F, N_i) \). The latter reduction equation follows similarly. To prove that \( \text{TM}(i(K)|_{\psi}(F) \) holds from these, we use Lemma 2: for any \( \psi' \), we have either \( \vdash i \xi_i \psi' \), in which case \( \text{TM}(i(K)|_{\psi}(F \psi') \) by the former equation, \( \vdash \psi' \rightarrow F \psi' \), in which case we apply the latter equation, or none of these, in which case \( i(K)\psi(F \psi') \) by definition. The binary case follows similarly. Intuitively, proving introduction rules for \( \text{TM}(\alpha) \) consists in checking that each possible reduction induces an equation in \( \text{TM}(\alpha) \).

The theorem for intro rules follows similarly, proven by mutual induction on \( |K| \) with a typing theorem for \( \{ [\cdot \cdot \cdot ]^K(-) \). Once we have defined a value-coherent \( i(K) \) relative to any \( \tau \) and well-formed \( K \) therein, it is straightforward to extend the construction in [3, §3] to build a type system which is closed under inductive types: \( \text{ind}(K) \equiv \text{ind}(K') \) type\(_{\text{pre}} \) \( \Psi \) with whenever \( K \equiv K' \) constrs \( \Psi \). We will omit this step, as it is technically involved and orthogonal to our contribution. As the open type judgments are defined by functionality, we have parameterized CTTs for free: for any \( \Gamma \gg K \equiv K' \) constrs \( \Psi \), we will have \( \Gamma \gg \text{ind}(K) \equiv \text{ind}(K') \) type\(_{\text{pre}} \) \( \Psi \).

For the remainder of Section 4, we will assume we are working in such a type system.
The first thing we want to do is define `values`. The operational semantics of recursive arguments.

In each clause eliminator. In general, this is an issue, and the proof goes through. The same technique works in the general case: for `x` and `y`.

For `y`, not commute with arbitrary functions. However, this is true up to an interpolating path: `coe` on glue, we thus wrap the naive solution in an `fcom` which adjusts the faces `x = 0` and `x = 1`, pushing `coe` through `F` on the left and `G` on the right. This corrects the coherence issue, and the proof goes through. The same technique works in the general case: for `0`-constructors, we push inside, and for higher constructors we ensure coherence by an `fcom` with a tube entry for each boundary constraint.

4.4 Elimination

To prove an elimination theorem, we will first have to specify the data needed by the eliminator. In general, this is an elimination list \( \mathcal{E} \) with a clause for each constructor:

\[
\mathcal{E} ::= \cdot | [\mathcal{E}, t : x_i : \gamma, \delta, \rho, R].
\]

In each clause `R`, we have access to dimension parameters \( x_i \), non-recursive arguments \( \gamma \), recursive arguments \( \delta \), and results \( \rho \) (with \( |\rho| = |\delta| \)) of recursive calls on variables in \( \delta \).
To express the types of recursive call results and coherence conditions, we define dependent instantiations of argument types and terms. For types, we set

\[
\{X\}_{a}(h.D; N) := D[N/h], \quad \{b:B \rightarrow c\}_{a}(h.D; N) := (b:B) \rightarrow \{c\}_{a}(h.D; \text{app}(N, b)).
\]

If \(h : A \Rightarrow D\) type\(_{\text{Kan}}\) \([\Psi]\), then \(b : \{b\}(A) \Rightarrow \{b\}_{a}(h.D; b)\) type\(_{\text{Kan}}\) \([\Psi]\). These types describe the results of calls to \text{elim} on recursive arguments, which are mediated by a dependent functorial action operator \text{func}. To express the coherence conditions on the clauses of an elimination list, we similarly need a dependent instantiation \(\{\theta,M\}_{h.D}(N_j; S_j)\) for terms. The term \(\{\theta,M\}_{h.D}(N_j; S_j)\) combines cases in \(E\) and results \(S_j\) of (func-mediated) recursive calls on terms \(N_j\) according to the shape of \(M\), producing the result of a (func-mediated) recursive call to \(\{\theta,M\}_{h.D}(N_j)\). We refer the reader to our preprint for the definition. Using these instantiation functions, we can define the equality \(E \equiv E' : K \rightarrow h.D\) \([\Psi]\) of elimination lists mapping from \text{ind}(K) into \(h : \text{ind}(K) \Rightarrow D\) type\(_{\text{Kan}}\) \([\Psi]\). For each constructor \(\ell\) with corresponding case \(\gamma_t : \gamma, \delta, \rho.R\), we ask that

\[
\gamma : \Gamma, \delta : \{b_j\}_{a}(\text{ind}(K)), \rho : \{b_j\}_{a}(h.D; \delta_j) \Rightarrow R \in D[\text{intro}^\gamma_{K,a}(\gamma; \delta)/h] \quad [\Psi, \mathcal{X}]
\]

and that moreover \(R \simeq \{\theta,M\}_{h.D}(\delta; \rho)\) under \(\xi_k\) for each \(k\). For the example of the pushout, this gives exactly the principle described in Section 2. To give an example with a recursive constructor, the truncation eliminator will require \(a : A \Rightarrow R_{\text{pt}} \in D[\text{pt}(a)/h] \quad [\Psi]\) and \(t_0, t_1 : |\mathcal{A}|, r_0 : D[t_0/h], r_1 : D[t_1/h] \Rightarrow R_{\text{path}} \in D[\text{path}^x(t_0; t_1)/h] \quad [\Psi, x]\) satisfying \(R_{\text{path}} \simeq t_0\) under \(x = 0\) and \(R_{\text{path}} \simeq t_1\) under \(x = 1\).

The operational semantics of \(\text{elim}_{h.D}(E)\) are straightforward. On an intro, it steps to the appropriate clause in \(E\), calling (the functorial action of) itself on the recursive arguments. On an \text{fcom}, it steps to a composition in the target type. Specifically, it steps to a heterogeneous composition \text{com}, a generalization of \text{fcom} derivable from the Kan operations where the type can vary with the tube (see [3, Theorem 44]). Each reduction rule is coherent in the target type, inducing a corresponding \(\beta\)-equation—this follows from the coherence conditions on the elimination data. From these equations, we can deduce a rule equating \text{elims} applied to equal arguments and elimination lists.

### 4.5 Canonicity

Our goal has been to give a semantics of CITs with a canonicity theorem. At this point, we can claim to have given a semantics for CITs, so let us return to canonicity. By Lemma 22, \(M \simeq M' \in \text{ind}(K)\) \([\Psi]\) implies that \(M \Downarrow V\) and \(M' \Downarrow V'\) with \(i(K)(V, V')\). (We also have \(M \simeq M' \in \text{ind}(K)\) \([\Psi]\); see [3, Lemma 38]). Thus, any term in a CIT reduces either to an intro or an \text{fcom} in that type.

For the zero-dimensional case, we can even get a stronger guarantee: any term \(M \in \text{ind}(K)\) \([\_]\) evaluates to a zero-dimensional intro in that type. This is a consequence of the validity condition we imposed on tube constraints back in Section 3.3. The assumption that \(\Xi\) is valid is a conservative approximation of the assumption that for every closing substitution \(\psi : \cdot \Rightarrow \Psi\), there is some \(\xi \in \Xi\) with \(\models \xi \psi\). Thus, any dimension-closed \text{fcom} or higher intro term reduces, leaving as values only the zero-dimensional constructor terms. This means that we can observe some interesting information by evaluating (zero-dimensional) points in CITs. For example, we can determine whether a point in \(A \Pi^{xG}_C B\) is on the left or right, and we can evaluate a point in \(|A|\) to extract an element of \(A\). Relatedly, Huber has shown that one can extract some \(M' \in A\) \([\Psi]\) from \(M \in |A|\) \([\Psi]\) even without a validity condition (and for \(\Psi\) non-empty), but in this case one may not have \(M \simeq \text{pt}(M') \in |A|\) \([\Psi]\) [21, Theorem 5.1].
This corresponds to the homotopical interpretation of \( \text{Id}_A \) as factorization of the diagonal \( \Delta : A \to A \times A \) [6]. By contrast, the path type \( \text{Path}_A(M_0, M_1) \), whose elements are abstracted terms \( \langle x \rangle M \) in \( A \) with \( M \langle 0/x \rangle = M_0 \in A \) \( \Psi \) and \( M \langle 1/x \rangle = M_1 \in A \) \( \Psi \), is not an identity type. While we can write equivalent rules of \( \text{refl} \) and \( J \) for \( \text{Path} \), they do not satisfy the expected \( \beta \)-rule exactly, only up to a path. This is related to the failure of regularity of \( h\text{com} \) and \( \text{coe} \) [14, 15]. A different type—here, \( \text{fib} \)—is thus necessary to make \( \text{CHITT} \) a model of \( \text{ITT} \).

With CTIs, we introduced formal \( \text{fcom} \) values in order to account for composition of \( n \)-constructors, while coercion was implemented using the Kan structure of component types. Intuitively, this kind of implementation of \( \text{coe} \) is possible because the construction of \( \text{ind}(K) \) \( \text{type}_{\text{kan}} \) \( \Psi \) is uniform in \( \Psi \). For indexed inductive types, the calculus shifts: the definition of \( b : B \supset \text{fib}(A; B; a.F; b) \) \( \text{type}_{\text{kan}} \) \( \Psi \) is still uniform in \( \Psi \), but not in \( B \). It thus becomes necessary to introduce formal coercions, \( \text{fcoes} \), to transport between fibers \( b : B \) of the family. The operation \( \text{coe} \) will be defined as a combination of \( \text{fcoes} \) and \( \text{total space} \).
coercions, tcoes, which transport between aspects of the arguments $A, B$, and $a.F$ using the Kan structure of $A$ and $B$. The term $\text{tcoe}_{x,T}^{r,r'}(O)$ will coerce $O \in \text{fib}(A; B; a.F; P(r/x))$ along $P$ to obtain a term in $\text{fib}(A; B; a.F; P(r'/x))$; its operational semantics is defined in Figure 5. Much like fcom, it steps to $O$ when $r = r'$ and is otherwise a value.

**Definition 9.** Let $A, B \text{ type}_{\text{Kan}}[\Psi]$ and $a : A \Rightarrow F \in B[\Psi]$ be given. We define a $\Psi$-relation $\text{fib}(A, B, a. F)$ over $B$, a function taking every $\psi : \Psi' \rightarrow \Psi$ and $N \in B\psi[\Psi']$ to a $\Psi$-relation $\text{fib}(A, B, a. F)[N, 1]$ to be least such that

1. $\text{fib}(A, B, a. F)[N]_{\psi}(\text{refl}(M), \text{refl}(M'))$ when
   \[ M \equiv M' \in A\psi\psi'[\Psi'], \]
2. $\text{fib}(A, B, a. F)[N]_{\psi}(V, V')$ when $\text{FCOM}(\text{fib}(A, B, a. F)[N])_{\psi}(V, V')$, \[ P \equiv P' \in B\psi\psi'[\Psi'], \]
3. $\text{fib}(A, B, a. F)[N]_{\psi}(\text{tcoe}_{x,T}^{r,r'}(O), \text{tcoe}_{x,T}^{r,r'}(O))$ when $r \neq r'$ and \[ P \eqv P' \in B\psi\psi'[\Psi'], \]

It is straightforward to check that each $\text{fib}(A, B, a. F)[N]$ is value-coherent. Henceforth, we assume we are in a cubical type system where $\text{fib}(A; B; a.F; N)$ names $\text{fib}(A; B; a.F)[N]$ for all $N$. For convenience, we will group the first three arguments to fib, writing $\text{fib}(\cdot; \cdot; \cdot; N)$.

To make up the difference between fcoe and coe, we define the total space coercion tcoe. Like coe in CITs, this is an eager operator which pushes into each constructor of the fiber type. Similarly, it uses the Kan structure of the $A$ and $B$, as well as the formal Kan structure to make necessary adjustments. The operational semantics is given in Figure 5. Writing $T := (A; B; a.F)$, we can derive the following typing theorem for tcoe:

**Lemma 10.** Let $\psi : (\Psi', x) \rightarrow \Psi$. For $N \in B\psi(r/x)[\Psi']$ and $O \in \text{fib}(T\psi(r/x); N)[\Psi]$, we have $\text{tcoe}_{x,T}^{r,r'}(O) \in \text{fib}(T\psi(r'/x); \text{coe}_{x,T}^{r,r'}(N)) [\Psi]$ and $\text{tcoe}_{x,T}^{r,r'}(O) \Rightarrow O \in \text{fib}(T\psi(r/x); N)[\Psi]$.

Note that tcoe carries the index $N$ along via coe in $B$. To implement coe in fib, we use a tcoe to get to the desired $A, B, a.F$, then follow up with fcoe to reach the target index:

$$
\text{fib}(T; N)(r/x) \xrightarrow{\text{tcoe}_{x,T}^{r,r'}} \text{fib}(T(r'/x); \text{coe}_{x,B}^{r,r'}(N(r/x)))
$$

The eliminator, which we suggestively name J, takes a clause $a.R$ for the refl case and satisfies the following typing rules:

**Lemma 11.** Let a family $b : B, h : \text{fib}(A; B; a.F; b) \Rightarrow D \text{ type}_{\text{Kan}}[\Psi]$ be given. If we have $a : A \Rightarrow R \in D[F, \text{refl}(a)/b, h][\Psi]$, then $b : B, h : \text{fib}(A; B; a.F; b) \Rightarrow J_{h, D, b}(h ; a.R) \in D[F, \text{refl}(a)/b, h][\Psi]$. And $a : A \Rightarrow \text{fib}(r; a. R) \Rightarrow R \in D[F, \text{refl}(a)/b, h][\Psi]$.

Like elim for CITs, J takes fcoms and fcoes to the corresponding Kan operations in $D$.

On refl terms, J steps to the provided $R$. The reduction equation in Lemma 11 thus follows immediately from Lemma 3.

---

1 We require that $f(A, B, a.F)[N]_{\psi'} = f(A, B, a.F)[N_{\psi'}]$ for every $\psi'$ and that $f(A, B, a.F)[\psi[-]$ respects equality in $B$. 
Related and future work

With regard to cubical inductive types, we can group the related work into three broad categories: schemata for HITs in HoTT and related theories, semantics of HITs, and the translation of HITs into cubical type theories.

In the first category, the earliest syntactic class is Sojakova’s $W$-quotients [33], which allow for some recursive structure but do not directly account for types like truncations. Sojakova showed that these types are homotopy-initial algebras, building on work on ordinary inductive types in HoTT [5]. More recently, Basold et al. [7], Dybjer and Moeneclaey [19], and Kaposi and Kovács [24] have given schemata which closely resemble ours, being described by a grammar or type of argument types and terms. The first accommodates 1-constructors, the second 2-constructors, and the third $n$-constructors as well as higher inductive-inductive types. Other work has focused on encoding complex HITs in terms of simpler ones [37, 26, 31], but it is not clear that this is possible for such HITs as general localizations. In comparison with this work, we benefit substantially from the cubical setting, which allows us to handle $n$-constructors uniformly and to give a canonicity theorem. While Kaposi and Kovács do account for $n$-dimensional constructors, elimination principles become increasingly complex with higher dimensionality. This seems to be an unavoidable issue in HoTT: there, $\beta$-rules for path constructors only hold up to identity (see [36, §6.2]), which means that stating eliminators requires more and more path algebra as constructors refer to each other.

In the second category, Lumsdaine and Shulman [28] define a model-categorical class of HITs which includes our examples but has no obvious syntactic equivalent. Due to size issues with fibrant replacement, their parameterized HITs do not live in the same universe as their parameters. In our setting, the role of replacement is played by $\text{fcom}$ and $\text{fcoe}$; our more fine-grained control seems to let us to sidestep the issue. For related reasons, they are not able to use composition in boundary terms.

Finally, we have the cubical type theories. The CCHM theory and AHF semantics each include a circle type, and CCHM moreover sketches spheres and truncation. Huber [22] has proven canonicity for CCHM, while the AHF semantics satisfies canonicity by definition. More recently, Coquand et al. have extended the CCHM theory with further examples (such as pushouts and two approaches to the torus), sketched a schema, and proven consistency with a semantics in cubical sets [16].

With regard to the $\text{fib}$ type and its raison d’être—defining $\text{Id}$—the original work is by Swan, who defined an identity type in a category equivalent to the substructural BCH model [34]. Swan’s $\text{Id}$ is similar to ours, being defined as a restricted fibrant replacement, but it is not clear how to adapt the construction to structural cubical type theories or to general indexed inductive types. Drawing inspiration from Swan, the CCHM theory defines identities as paths with labelled degeneracies that make it possible to distinguish reflexive paths. Bezem et al. have extended the BCH model with an identity type using another related construction, this one based on a cofibration-trivial fibration factorization [9]. Again, it is unclear whether these approaches generalize to indexed inductive types. Awodey has given a cubical model in which $\text{Path}$ and $\text{Id}$ coincide, but which is not known to support a univalent universe [4].

We are currently extending RedPRL [35], a mechanized proof theory for the AHF semantics, with our schema for CITs. There remains much room for generalization: besides the indexed CITs we hint at in Section 5, there are the higher inductive-inductive types of Kaposi and Kovács, for which there is at present no known univalent semantics. Even for ordinary CITs, there are holes to be filled: for example, argument terms defined by induction
on positive types (see [12, §6.6]) or argument types using the path types of the inductive.

References

15. Thierry Coquand. Re: [HoTT] a cubical type theory. Mailing list post. groups.google.com/d/msg/homotopytypetheory/oXQe5u_Mmtk/3HEDx5g5uq4J, May 2015.


