Higher Inductive Types in Cubical Computational Type Theory

Evan Cavallo  
Carnegie Mellon University  
cecavallo@cs.cmu.edu

Robert Harper  
Carnegie Mellon University  
rwh@cs.cmu.edu

Abstract
In homotopy type theory (HoTT), higher inductive types provide a means of defining and reasoning about higher-dimensional objects such as circles and tori. The formulation of a schema for such types remains a matter of current research. We investigate the question in the context of cubical type theory, where the homotopical structure implicit in HoTT is made explicit in the judgmental apparatus. Within the computational cubical type system framework of Angiuli et al., we implement a class we call cubical inductive types, which includes n-truncations, W-quotients, and localizations.

We suggest an extension to indexed inductive types by defining an example, a homotopy fiber type. From this we derive an identity type, making our theory a model of Martin-Löf type theory. Using Angiuli et al.’s implementation of univalence, we obtain a computational interpretation of HoTT with a general class of higher inductive types. This interpretation admits a canonicity theorem: any zero-dimensional element of a cubical inductive type evaluates to a constructor.

1 Introduction
Formal Martin-Löf type theory (MLTT) [23, §2], a logic for constructive mathematics, includes identity types; an element of the identity type IdA(M, N) expresses that M, N : A are in a sense interchangeable. By definition, IdA(−, −) is the least reflexive “relation” on A, being generated by a constructor a : A → refl(a) : IdA(a, a). But IdA(−, −) is not merely a relation: the groupoid model of Hofmann and Streicher [16] and subsequent homotopical models of Awodey and Warren [9] and Voevodsky [19] showed that IdA(−, −) can be interpreted as a sort of proof-relevant relation whose elements are means of identifying M and N. The homotopical models suggest a view of elements P : IdA(M, N) as paths between M and N, and of A itself as a space. Homotopy type theory (HoTT) [30] enforces interpretations of this kind by postulating the univalence axiom and higher inductive types (HITs).1

Our focus is on the latter, which enable the definition of higher-dimensional objects as inductively generated by some combinatorial presentation of identification structure. With these in hand, one can reason type-theoretically about objects like circles and tori, a practice which is called synthetic homotopy theory. However, despite the wealth of intuitive examples, a precise definition of “higher inductive type” remains elusive.

We pursue such a definition in the Computational Higher Type Theory (CITT) of Angiuli et al. [4–6], one of a class of cubical type theories which have arisen from attempts to make HoTT constructive. Sparked by the cubical model of Bezem, Coquand, and Huber [11] (BCH) and pursued by a number of groups [3, 5, 12, 14], cubical type theories bake higher structure into their judgmental apparatus rather than exposing it via axioms on an identity type. The cubical theory of Cohen, Coquand, Huber, and Mörtberg [14] (CCHM) was the first univalent type theory with a canonicity result [17] stating that any closed term can be evaluated to some canonical form: for example, any closed term of boolean type is equal either to true or to false. This is a property that HoTT does not enjoy: the theory provides no way of evaluating the Id-eliminator J on identifications created by the new axioms. In addition to providing computational interpretations of HoTT, the cubical type theories are of interest in their own right, having practical advantages for synthetic homotopy theory first observed by Brunerie and Licata [12, 21].

CITT, as distinguished from the other cubical approaches, is a framework for Nuprl-style computational type theories [1], in which types are behavioral specifications on terms in an untyped programming language. Such type theories are particularly well-suited to obtaining canonicity results: essentially by definition, a well-typed term evaluates to a canonical form in its type. The most recent iteration includes a treatment of univalence along with every type former of MLTT except Id (instead supporting an analogous Path type) [6].

Our contribution is to extend CITT with a schema for cubical inductive types, giving a precise adaptation of the informal notion of HIT to cubical type theory. This enables the pursuit of synthetic homotopy theory in a genuinely computational setting. Besides interpreting the majority of HITs postulated in the HoTT Book [30, §6], cubical inductive types validate more useful equalities and more easily accommodate generators of arbitrary dimensionality.

We further define a homotopy fiber type, gesturing towards a theory of indexed (cubical) inductive types. This type can be used to construct any (zero-dimensional) indexed inductive type with only non-recursive constructors. In particular, we define Id, the indexed inductive type generated by refl, making CITT into a full computational interpretation of HoTT.

In this paper, we present the central definitions and proof methodology. The full technical details can be found in our preprint [13].

2 Cubical inductive types
Our first contribution is the development of a schema for cubical inductive types (CITs) and an implementation of the types so described. CITs subsume many HITs historically proposed for HoTT, but constructors introduce higher cubes rather than elements of identity types, in keeping with the cubical nature of CITT.

As a first example, the circle S1 is a CIT generated by a 0-dimensional “point” constructor base and a 1-dimensional “line” constructor loop, where x is a dimension parameter. We think of dimension parameters as ranging over the unit interval [0, 1], so that the constructor loop traces out a line as x varies from 0 to 1. The endpoints of a 1-constructor are attached to existing terms: here, the endpoints loop and loop of the loop constructor each reduce to base.

Observe that loop is not the only line in S1: we can draw a line which goes around loop twice, or one which goes around loop in the other direction.

1We reserve the term HoTT for the theory defined in [30].
3 Computational higher type theory

We define our schema in Angiuli et al.’s CHTT. This is a framework for specifying computational type theories, systems of types which name partial equivalence relations (PERs) on the values of an untyped language. CHTT generalizes ordinary computational type theory on the level of terms and of types: we have a cubical language in which terms contain dimension names, and types name collections of PERs stratified by dimensionality.

3.1 Programming language

Dimensionality is expressed via dimension terms. A dimension term is either 0, 1, or a formal dimension name; we write \( x, y, z \) for names and \( r, s \) for terms. A dimension context is a finite set \( \Psi = \{x_1, \ldots, x_n\} \) of names. We write \( r \dim \{\Psi\} \) to mean that \( r \in \Psi \cup \{0, 1\} \).

Dimension terms can occur in ordinary terms, for which we write \( M, N, O \), and so on. We write \( M \tm \{\Psi\} \) to mean that \( \text{FD}(M) \subseteq \Psi \), where \( \text{FD}(M) \) is the set of names occurring free in \( M \). We think of a term \( M \tm \{\Psi\} \) as varying over \( \{0, 1\}^{|\Psi|} \), thus describing a \( \{0, 1\} \)-cube.

A dimension substitution \( \psi : \Psi' \to \Psi \) takes names \( x \in \Psi \) to dimension terms \( \psi(x) \dim \{\Psi'\} \). For any \( M \tm \{\Psi\} \), we have a term \( M\psi \tm \{\Psi'\} \) obtained by replacing each \( x \) in \( M \) with \( \psi(x) \). We call \( M\psi \) an aspect of \( M \).

The operational semantics is defined by judgments \( M \ap \psi \to M' \psi \) and \( M \tm \{\Psi\} \). These apply to terms which are closed, i.e., contain no term variables, but which may contain dimension names; our goal, after all, is to compute with higher-dimensional objects. We say that \( M \tm V \) when \( M \to V \) and \( V \tm \{\Psi\} \). The operational semantics is unstable in the sense that \( M \ap \psi \to M' \psi \) does not imply \( M\psi \tm \{\Psi\} \to M'\psi \tm \{\Psi\} \). For example, loop\(^x\) is a value, but loop\(^y\) is base.

3.2 Cubical type systems

A cubical type system for such a language is a four-place relation \( \tau(\Psi, A_0, B_0, \varphi) \) relating contexts \( \Psi \), values \( A_0, B_0 \tm \{\Psi\} \), and PERs \( \varphi \) on values \( V, V' \) with \( \text{FD}(V, V') \subseteq \Psi \). When \( \tau(\Psi, A_0, B_0, \varphi) \) holds, we say that \( A_0 \) and \( B_0 \) are equal canonical pretypes at \( \varphi \) denoting \( \varphi \).

We say that terms \( A, B \), not necessarily values, are equal pretypes at \( \Psi \) when they coherently evaluate to equal canonical pretypes. This condition requires, among other things, that for every \( \psi : \Psi' \to \Psi \) there exist values \( A_0, B_0 \) and some PER \( \alpha\psi \) such that \( A_0 \equiv \alpha\psi (B_0) \). We call such a family \( \alpha = (\alpha\psi)_{\psi \in \Psi} \) of relations a \( \Psi \)-relation and write \([A] \) for the \( \Psi \)-PER associated to a pretype \( A \tm \{\Psi\} \). If \( \alpha \) is a \( \Psi \)-relation and \( \psi : \Psi' \to \Psi \), we write \( \alpha\psi \) for the \( \Psi' \)-relation defined by \( (\alpha\psi)\varphi = \alpha\varphi \).

Any value \( \varphi \)-relation extends to a term \( \Psi \)-relation \( \text{TM}(\alpha) \) relating terms which coherently evaluate to values in \( \alpha \). The \( \Psi \)-relation \( \text{TM}(\alpha) \) is always stable; where a \( \Psi \)-relation \( \beta \) is stable when \( \beta(M, M') \equiv \beta\varphi(M, M') \) for all \( \varphi, M, M' \). The \( \Psi \)-relations we use will typically not be stable, as the notion of valuehood itself is not stable. To introduce values in \( \text{TM}(\alpha) \), we have the following.

Lemma 3.1 (Introduction). Let \( \alpha \) be a value \( \Psi \)-PER. If for every \( \psi : \Psi' \to \Psi \), either \( \alpha\psi(M\psi, M'\psi) \) or \( \text{TM}(\alpha)\psi(M\psi, M'\psi) \), then \( \text{TM}(\alpha)(M, M') \).

\(^1\)The precise definition of “coherent evaluation” is somewhat technical and unenlightening, so we omit it and refer the reader the definitions of \( \text{FPY} \) and \( \text{TM} \) in [6].

\(^2\)For any \( \Psi \)-relation \( \beta \), we abbreviate \( \beta\varphi(M, M') \equiv \beta(M, M') \).
Higher Inductive Types in Cubical Computational Type Theory

First, we want to be able to compose cubes. This is easiest to see in the 1-dimensional case: suppose, as shown on the left in Figure 3, that we have two composable lines in a type \( A \), i.e., two terms \( P \in A \{ x \} \) and \( Q \in A \{ y \} \) such that \( P(1/x) \equiv Q(0/y) \in A \{ . \} \). Then we should be able to concatenate them and obtain a line \( S \), perhaps regarded as varying in \( x \), such that \( S(0/x) \equiv P(0/x) \in A \{ . \} \) and \( S(1/x) \equiv Q(1/y) \in A \{ . \} \). We more generally imagine gluing two \( n \)-cubes along a shared face. The ability to compose cubes in these and other ways is summed up by the hcom-\( Kan \)-conditions, which require the ability to adjust any cube along a tube.

To define tubes, we will first need to define constraints. A constraint is an unordered pair \( \xi = \{ r_0, r_1 \} \) of dimension terms; we write \( (r_0, r_1) \) as sugar for \( \{ r_0, r_1 \} \). We say \( \xi = \{ r_0, r_1 \} \) holds and write \( \models \xi \) if indeed \( r_0 = r_1 \). A constraint context \( \Xi \) is a list of constraints; we write \( \models \Xi \) when \( \models \xi \) for all \( \xi \in \Xi \).

The judgments \( A \perp \Xi \) for \( \Xi \) and \( \Xi \) hold when \( A \perp \Xi \) and \( A \perp \Xi \). The reasons for this restriction are discussed in Section 4.2.

We say \( M \in A \{ \Psi \} \) is a cap for \( \xi_1 \mapsto y.N_1, \ldots, \xi_n \mapsto y.N_n \) at \( r \) when \( M \in A \{ \Psi \} \) for each \( i \). Intuitively, we are asking that \( M \) fit into the tube at \( r \). The hcom-\( Kan \)-conditions require that, given such a tube and cap, we can adjust the cap to produce a term that fits at any other \( r' \). Syntactically, we ask that

\[
\text{hcom}_{A}^{r \mapsto r'}(M; \xi_1 \mapsto y.N_1) \in A \{ \Psi \}
\]

with \( \text{hcom}_{A}^{r \mapsto r'}(M; \xi_1 \mapsto y.N_1) \in A \{ \Psi \} \) for all \( i \). The hcom-\( Kan \)-conditions also require that "motionless" hcoms are degenerate, i.e., that \( \text{hcom}_{A}^{r \mapsto r'}(M; \xi_1 \mapsto y.N_1) \in A \{ \Psi \} \) for all \( i \). The name hcom stands for "homogeneous composition," we refer to the term above as a composite.

To see how this condition encompasses ordinary composition, we return to Figure 3. We can now define

\[
S := \text{hcom}_{A}^{0 \mapsto 1} \left( P, x = 0 \mapsto y.P(0/x) \right) \bigg| x = 1 \mapsto y.Q.
\]

Visually, we have the picture shown on the right: \( P \) is adjusted by a tube with a degenerate path \( y.P(0/x) \) at \( x = 0 \) and \( y.Q \) at \( x = 1 \).

In addition to composition, we want coercion. Suppose we have \( A \perp \Xi \). If we think of \( A \) as a type varying continuously in \( x \), then it stands to reason that any element \( M \in A \{ 0/x \} \{ \Psi \} \) can be traced through \( A \) to obtain some \( M' \in A \{ 1/x \} \{ \Psi \} \). The coercion-\( Kan \)-conditions require more generally that for every \( r, r' \), and \( M \in A \{ r/x \} \{ \Psi \} \), we have

\[
\text{coerce}_{A}^{r \mapsto r'}(M) \in A \{ r'/x \} \{ \Psi \},
\]

with \( \text{coerce}_{x.A}^{r \mapsto r'}(M) \in A \{ r'/x \} \{ \Psi \} \).
Formally stated, the hcom- and coe-Kan conditions require that the operators hcom and coe satisfy these properties at any aspect $A\triangleright\Psi$ of $A$. We say that pretypes $A\equiv A'$ typepre [$\Psi$] are equally hcom-Kan when they satisfy the hcom-Kan conditions and each $hcom_{A\triangleright\Psi}$ term is equal to the corresponding $hcom_{A'\triangleright\Psi}$ term in $A\triangleright\Psi$. Likewise, we say that $A\equiv A'$ typepre [$\Psi$] are equally coe-Kan when they satisfy the coe-Kan conditions with equal implementations of coe. When $A\equiv A'$ typepre [$\Psi$] are equally hcom- and coe-Kan, we say they are equally Kan and write $A\equiv A'$ typeKan [$\Psi$]. The open form $\Gamma\triangleright\Psi\triangleright\beta_{\Psi}hcom_{\Psi}K\triangleright\Theta\triangleright\Psi\triangleright\beta_{\Psi}$ to mean that $\Gamma\triangleright\Psi\triangleright\beta_{\Psi}hcom_{\Psi}K\triangleright\Theta\triangleright\Psi\triangleright\beta_{\Psi}$ and each pair of pretypes in $\Gamma, \Gamma'$ is equally Kan. Note that these are conditions on specific syntactic operators: ensuring $A$ is Kan means defining the language so that hcom$_A$ and coe$_{A,A}$ evaluate in a well-typed way.

In any Kan type, we can derive a “heterogeneous composition” operator com which constructs composites where the type also varies in the tube dimension. (Such an operator is taken as primitive in [3, 11, 14].) The definition and typing rule can be found in [6].

4 CITs in generality

We are now well-equipped to treat cubical inductive types in CITT. This task naturally decomposes into three parts.

In Section 4.1, we give our schema, which centers around a judgment $K$ constrs [$\Psi$] specifying well-formed lists of constructors. Here, the complication is the need to account for boundaries; as is typically done with recursive argument types, we restrict the allowable boundary terms to a fixed grammar.

In Section 4.2, we define the value $\Psi$-PER i($K$) generated by $K$. In this step, we must take care to ensure the inductive type will be Kan. As such, $i(K)$ consists not only of constructor terms, but also of formal composites.

In Section 4.3, we check that $i(K)$ has the properties we expect of an inductive type: the introduction forms construct elements in it, it is Kan, and there is an eliminator which maps out of it. In computational type theory, these properties are theorems, consequences of the operational semantics and the definition of $i(K)$.

4.1 The schema

The central judgment of our schema, $K \equiv K'$ constrs [$\Psi$], states that $K$ and $K'$ are equal lists of constructors. It is mutually inductively defined with a judgment $K + C \equiv C'$ constr [$\Psi$] asserting that $C$ and $C'$ are equal constructors in the context of a prefix $K$.

Per Figure 4, a constructor list is a list of labelled constructors, each well-formed over its prefix. We draw labels $\ell$ from some set $L$ and write $\ell \in K$ to mean that $\ell$ occurs in $K, K[\ell]$ for the constructor in $K$ labelled $\ell$, and $K_{<\ell}$ for the prefix preceding $\ell$.

\[
\begin{align*}
\text{K} & \equiv C' \text{ constrs } [\Psi] \\
\ell & \not\in K \\
K + C & \equiv C' \text{ constr } [\Psi] \\
[K, \ell : C] & \equiv [K', \ell : C'] \text{ constrs } [\Psi]
\end{align*}
\]

Figure 4. Constructor list well-formedness.

The substance is thus encapsulated in the constructor equality judgment, itself defined in terms of judgments $A \equiv A'$ atype [$\Psi$].

\[
\begin{align*}
\Theta & \equiv \Theta' \text{ actx } [\Psi], \text{ and } K; \Theta + m \equiv m' : A [\Psi]. \text{ These judgments form a small formal type theory of argument types and terms which can recursively reference the type being specified.}
\end{align*}
\]

Definition 4.1. Presupposing $K$ constrs [$\Psi$], we say $C$ and $C'$ are equal over $K$ and write $K + C \equiv C'$ constr [$\Psi$] when

\[
\begin{align*}
C &= (\Gamma ; y, \Theta; \xi_i, \xi_k \leftarrow y, \Theta; M_k), \quad C' = (\Gamma' ; y, \Theta'; \xi_i, \xi_k \leftarrow y, \Theta'; M_k')
\end{align*}
\]

where

1. $\Gamma \equiv \Gamma'$ typeKan [$\Psi$],
2. $y : \Gamma \triangleright\Theta \equiv \Theta'$ actx [$\Psi$],
3. $\text{FD}(\xi_k) \subseteq \xi_i$ and $\xi_k$ is valid if $|\xi_i| > 0$,
4. $y : \Gamma \triangleright K; \Theta + m_k \equiv m'_k : X [\Psi, \xi_i \mid \xi_k, \xi_i]$ for each $k, l$.

A constructor consists of a context $\Gamma$ listing the types of its non-recursive parameters, an argument context $\Theta$ listing the types of its recursive arguments (possibly dependent on $\Gamma$), dimension parameters $\xi_i$, a list of constraints $\xi_k$ on $\xi_i$ at which it should reduce, and boundary terms $M_k$ prescribing the reduction behavior at each constraint in a compatible way.

As is standard for ordinary inductive types, argument types are certain strictly positive type operators, enumerated by the judgment $A \equiv A'$ atype [$\Psi$] inductively defined in Figure 5. The type $X$ is the recursive reference to the inductive type; every argument type has the form $a : A \rightarrow b \equiv b'$ atype [$\Psi$] rather than $\Rightarrow, \equiv$, to emphasize this point, and we use blue text to distinguish components of the schema language, which have no operational semantics of their own. However, the argument type judgment does use the ordinary type judgment in its definition, and the open form $\Gamma \triangleright A \equiv A'$ atype [$\Psi$] by functionality: we say $y : \Gamma \triangleright A \equiv A'$ atype [$\Psi$] when $A\psi_1[M_i/y] \equiv A'\psi_1[M'_i/y]$ atype [$\Psi'$] for all $\psi : \Psi \rightarrow \Psi$ and $M_i \equiv M'_i$ in $\Gamma\psi[\Psi']$. Where this open form occurs in Figure 5, one should imagine it replaced by its definition. An argument context $\Theta$ is a list of argument types; the open form $\Gamma \triangleright \Theta \equiv \Theta'$ actx [$\Psi$] is similarly defined.

We must now account for boundary terms, which determine how a constructor steps when a constraint holds. Here, we must be able to refer to previous constructors and recursive arguments. We thus define argument terms by a judgment $K; \Theta + m \equiv m' : A [\Psi]$ parameterized by a list $K$ of previous constructors. This judgment is defined in Figure 6. Notice that we are inductively defining a judgment which is open in an argument context $\Theta$. In contrast,
Higher Inductive Types in Cubical Computational Type Theory

Let us return to some examples. We can specify the pushout
\[ \Pi(A; B; C; F; G) \]
with a list which has 0-constructors
\[ \text{left} : (A; a; \varnothing) \quad \text{right} : (B; b; \varnothing) \]
and a 1-constructor
\[ \text{glue} : (C; c; \varnothing; x) \]
\[ x = 0 \leftrightarrow c.\text{intro} \quad x = 1 \leftrightarrow c.\text{intro} \]
(We use \( \varnothing \) to denote empty lists.) We can specify \( |A| \) as having a 0-constructor \( pt : (A; a; \varnothing; \varnothing) \) and a 1-constructor
\[ \text{path} : (\varnothing; \varnothing; (X, X); x) \]
\[ x = 0 \leftrightarrow \varnothing.\text{left}(t_0, t_1) \quad x = 1 \leftrightarrow \varnothing.\text{right}(t_0, t_1) \]

as in ordinary W-types, and a 1-constructor
\[ \text{cell} : (C; c; (B[I/c]\langle a \rangle) \rightarrow X, B[I/c]\langle a \rangle) \rightarrow X) ; x : \mathcal{T} \]
where
\[ \mathcal{T} : x = 0 \leftrightarrow c(g_0, g_1).\text{intro} \quad x = 1 \leftrightarrow c(g_0, g_1).\text{intro} \]

The 1-constructor is thus indexed by \( C \) and has sup terms as its endpoints, their non-recursive part determined by \( F, G \) and recursive part determined by a pair of recursive arguments to cell.

Further examples are given in [13, §6], including higher truncations and localization.

4.2 The \( \Psi \)-PER \( \iota(K) \)

We can now specify CITs, but we still need to construct them. For each \( K \) constrains \( \Psi \), we thus define a value \( \Psi \)-PER \( \iota(K) \) generated by \( K \).

An fcom is a free or formal composite. Recall that in a Kan type \( A \), a composite \( hcom_{A}^{r \leftarrow r'}(M; \xi \leftrightarrow y, N_i) \) is required to satisfy two boundary conditions: it must be equal to \( N_i(r'/y) \) or \( M(r/y) \). The operator fcom satisfies these constraints in the most simple-minded way:

\[ fcom_{A}^{r \leftarrow r'}(M; \xi \leftrightarrow y, N_i) \rightarrow N_i(r'/y) \quad \text{when} \quad \xi \text{ is a free variable} \]

This is the approach we take for CITs, where we cannot account for composition with intro terms alone.

An intro term takes the form \( \text{intro}^{\ell}(\tilde{P}_n; \tilde{N}_j) \), new annotations being necessary for the operational semantics: the constructor data \( C \) and the prefix \( K' \) occurring prior to \( \ell \) in \( K \). To use these annotations and to formulate the intro typing rule, we need a way of interpreting argument types and terms as real terms. For types,
we define a term $\bar{\text{A}}(\bar{A})$ interpreting $A$ at a given instantiation $A$ of $\bar{X}$:

\[
\{\bar{X}\}(\bar{A}) := A \quad \{\bar{b}:B \rightarrow \bar{c}(\bar{A})\} := \{\bar{b}:B \rightarrow \bar{c}(\bar{A})\}.
\]

We can similarly define the $\Psi$-relation interpretation $\bar{\text{A}}(\bar{A})$ of $A$ at a $\Psi$-relation $\alpha$. For terms, we define in Figure 8 the interpretation $\bar{\{\bar{\theta, M}\}} \Psi \bar{N_r}$ of a term $M$ in free argument variables $\theta$, which takes the prefix $\Psi$ and instantiations $\bar{N_r}$ for the argument variables.

We can now explain the operational semantics of intro: When a constraint $\xi_k \rightarrow \bar{N_r}$ is satisfied, it steps to the instantiation of the corresponding boundary term. Otherwise, it is a value.

We want $\bar{\text{i}}(\bar{K})$ to be the least value $\Psi$-relation closed under fcom and intro values. We thus define operators Fcom and $\text{Int}_\Psi$, $\text{Intro}_\Psi$ on value $\Psi$-relations and take $\bar{\text{i}}(\bar{K})$ to be their least fixed-point.

**Definition 4.2.** For any $\Psi$-relation $\alpha$, we say that $\text{Fcom}(\alpha)(V, V')$ holds when $V = \text{fcom}^\rightarrow\rightarrow (M; \xi_i \mapsto y.N_j)$, $V' = \text{fcom}^\rightarrow\rightarrow (M'; \xi_i \mapsto y.N'_j)$ where

1. $\xi_i$ is valid, $\not\models \xi_i$ for all $i$, and $r \neq r'$;
2. $\xi_i \mapsto y.N_j$ and $\xi_i \mapsto y.N'_j$ are equal tubes in $\text{Tm}(\psi)$ with equal caps $\text{Tm}(\psi)(M M')$ at $r$.

**Definition 4.3.** Let $\ell \in \Psi$. For any $\Psi$-relation $\alpha$, we say that $\text{Intro}_\Psi(\alpha)(V, V')$ holds when

\[
V = \text{intro}_\Psi(\alpha)(V, V'), \quad V' = \text{intro}_\Psi(\alpha)(V', V'')
\]

where

1. $\Psi(\ell) = (\bar{\Gamma}; \bar{\theta, \Xi, \bar{\xi}_k \rightarrow y.\bar{\theta, M_k}})$ with $\Theta = \bar{p}_j : \bar{b}_j$,
2. $\Psi(\ell) \prec \ell \equiv \Psi(\ell) \equiv \Psi(\ell)$ constrs $[\Psi']$,
3. $\Psi(\ell) \prec \ell \equiv \Psi(\ell) \equiv \Psi(\ell)$ constrs $[\Psi']$,
4. $\not\models \xi_k \rightarrow \bar{N_r}$ for all $k$,
5. $\bar{P}_n = \bar{P}'_n \in \Gamma \{\Psi\}$.

6. $\{\bar{b}_j[\bar{P}_n/y]\}(\alpha\psi)(N_j, N'_j)$ for all $j$.

We define $T_K(\alpha) := \text{Fcom}(\alpha) \cup \cup_{\ell \in \Psi} \text{Intro}_\Psi, \ell$ and take $\bar{\text{i}}(\bar{K})$ to be the least fixed-point of $T_K$. With this definition in hand, we can construct a cubical type system $\tau$ with pretypes $\text{ind}(\bar{K})$ naming $\bar{\text{i}}(\bar{K})$ for each $\bar{K}$ constrs $[\Psi]$. The details of this construction are largely orthogonal to our contribution, so we refer the reader to Angiuli et al. [6] and simply assume we have such a type system. As the open type judgments are given by functionality, we get parameterized CITs for free: for any $\Gamma \gg \bar{K}$ constrs $[\Psi]$, we will have $\Gamma \gg \text{ind}(\bar{K})$ type pre $[\Psi]$.

**4.3 Rules for CITs**

We have defined $\bar{\text{i}}(\bar{K})$ by its universal property in the category of $\Psi$-relations and inclusions. Now we want to situate it in the category of Kan types and terms. In a formal type theory, these properties are postulated via rules, but here they are theorems about $\bar{\text{i}}(\bar{K})$ and $\text{ind}(\bar{K})$.

**4.3.1 Introduction and canonicity**

The introduction rules extend the closure properties of $\bar{\text{i}}(\bar{K})$ to $\text{Tm}(\bar{\text{i}}(\bar{K}))$. We also have boundary rules, which state that the reduction rules for fcom and intro induce equations in $\text{ind}(\bar{K})$.

**Proposition 4.4** (fcom boundary and introduction). If

1. $\xi_i$ is valid,
2. $\xi_i \mapsto y.N_j$ and $\xi_i \mapsto y.N'_j$ are equal tubes in $\text{ind}(\bar{K}\psi)$ with equal caps $M = M'$ at $r$,

then

a. $\text{fcom}^\rightarrow\rightarrow (M; \xi_i \mapsto y.N_j) \equiv N_j(r/y) \in \text{ind}(\bar{K}\psi) \{\Psi' \mid \xi_i\}$

for each $i$,

b. $\text{fcom}^\rightarrow\rightarrow (M; \xi_i \mapsto y.N_j) \equiv M \in \text{ind}(\bar{K}\psi) \{\Psi'\}$

c. $\text{fcom}^\rightarrow\rightarrow (M; \xi_i \mapsto y.N_j)$ and $\text{fcom}^\rightarrow\rightarrow (M'; \xi_i \mapsto y.N'_j)$ are equal in $\text{ind}(\bar{K}\psi)$ at $\Psi$.

**Proposition 4.5** (intro boundary and introduction). Let $\ell \in \Psi$. If

1. $\Psi(\ell) = (\bar{\Gamma}; \bar{\theta, \Xi, \bar{\xi}_k \rightarrow y.\bar{\theta, M_k}})$ with $\Theta = \bar{p}_j : \bar{b}_j$,
2. $\Psi(\ell) \prec \ell \equiv \Psi(\ell) \equiv \Psi(\ell)$ constrs $[\Psi']$,
3. $\Psi(\ell) \prec \ell \equiv \Psi(\ell) \equiv \Psi(\ell)$ constrs $[\Psi']$,
4. $\bar{P}_n = \bar{P}'_n \in \Gamma \{\Psi\}$,
5. $\{\bar{b}_j[\bar{P}_n/y]\}(\text{ind}(\bar{K}\psi))(N_j, N'_j)$ for all $j$.

then

a. $\text{intro}_\Psi(\alpha)(\bar{P}_n; N_j) \equiv N_j(\bar{P}_n; N_j)$ for each $j$,

b. $\text{intro}_\Psi(\alpha)(\bar{P}_n; N_j) \equiv \text{intro}_\Psi(\alpha)(\bar{P}'_n; N'_j) \equiv \text{ind}(\bar{K}\psi) \{\Psi'\}$

The boundary equations can be proven using Lemma 3.2. With these in hand, the binary introduction rules follow from Lemma 3.1. In each case, what we need is that the terms evaluate coherently, meaning that any interleaving of dimension substitution and evaluation gives the same result up to $\text{Tm}(\bar{\text{i}}(\bar{K}))$. In the fcom case, this follows by the compatibility conditions on the tube and cap; in the intro case, by the similar conditions on boundary terms in $\Psi$.

The binary introduction rules directly imply a key result:

**Corollary 4.6.** $\bar{\text{i}}(\bar{K})$ is value-coherent, and thus $\text{ind}(\bar{K})$ type pre $[\Psi]$.
\[
\begin{align*}
\{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j) & := N_j \\
\{\emptyset.\text{intro}^{\pi'}_f(\overline{P_n}; \overline{N}_j)\}_K^{\pi'}(\overline{N}_j) & := \text{intro}^{\pi'}_{\xi, f; K}(\overline{P_n}; \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)) \\
\{\emptyset.\text{fcom}^{\pi'-r'}(M; \xi, \zeta_j \mapsto y; \overline{N}_j)\}_K^{\pi'}(\overline{N}_j) & := \text{fcom}^{\pi'-r'}(\{\emptyset.M\}_K^{\pi'}(\overline{N}_j); \xi, \zeta_j \mapsto y.\{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)) \\
\{\emptyset.\emptyset\}_K^{\pi', E}(\overline{N}_j; \overline{S}_j) & := S_j \\
\{\emptyset.\text{intro}^{\pi', E}_f(\overline{P_n}; \overline{N}_j)\}_K^{\pi', E}(\overline{N}_j; \overline{S}_j) & := R(\overline{r}_1, \overline{s}_1)(\overline{P_n}; \delta_{\{\emptyset.\emptyset\}_K^{\pi', E}(\overline{N}_j)})(\overline{N}_j) \\
\{\emptyset.\text{fcom}^{\pi'-r'}(M; \xi, \zeta_j \mapsto y; \overline{N}_j)\}_K^{\pi', E}(\overline{N}_j; \overline{S}_j) & := \text{fcom}^{\pi'-r'}(\{\emptyset.M\}_K^{\pi', E}(\overline{N}_j); \xi, \zeta_j \mapsto y.\{\emptyset.\emptyset\}_K^{\pi', E}(\overline{N}_j; \overline{S}_j))
\end{align*}
\]

Figure 8. Argument term instantiation and dependent argument term instantiation. We omit the definitions for \(\lambda\) and \(\text{app}\).

\[
\begin{align*}
\text{mcoe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)} \colon & =: \cdot \\
\text{mcoe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)}(\overline{M}_n; \overline{M}) := \left(\text{mcoe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{M}_n)}(\overline{M})\right)
\end{align*}
\]

\[
\begin{align*}
(\forall i \not\vdash \xi_j) \\
\text{coe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)}(\text{fcom}^{\pi'-r'}(M; \xi, \zeta_j \mapsto y; \overline{N}_j)) := \text{fcom}^{\pi'-r'}(\text{coe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)}(M); \xi, \zeta_j \mapsto y.\text{coe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)}(\overline{N}_j)) \\
\mathcal{K}[\ell] = (\Gamma; g.\Theta; \emptyset, \emptyset) \\
\text{\Theta} = p_{\pi'} : b_{\pi'} \\
(\forall s, j) \overline{N}_j = \text{coe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)} \left(\{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)\right) \\
\text{coe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)} \left(\text{intro}^{\pi', E}_f(\overline{P_n}; \overline{N}_j)\right) := \text{intro}^{\pi', E}_f(\overline{P_n}; \overline{N}_j) \\
(\forall s, j) \overline{N}_j = \text{coe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)} \left(\{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)\right) \\
\text{coe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)} \left(\text{intro}^{\pi', E}_f(\overline{P_n}; \overline{N}_j)\right) := \text{fcom}^{\pi'-r'}(\text{intro}^{\pi', E}_f(\overline{P_n}; \overline{N}_j) \overline{r}'_{/z}. \overline{\mathcal{K}}[\ell] r'_{/z} = \overline{N}_j)
\end{align*}
\]

Figure 9. Operational semantics of \(\text{coe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)}\) on values. The meta-operation \(\text{mcoe}\) extends coercion from Kan types to Kan contexts.

At this point, we can state our canonicity result in its strongest form. The proof relies on the fact that for a value-coherent \(\Psi\)-PER \(\alpha\), \(\text{Tm}(\alpha)\) \(\subseteq (M, M)\) and \(M \not\forall V\) imply \(\text{Tm}(\alpha)\) \(\subseteq (M, V)\) [6, Lemma 37].

**Proposition 4.7.** Let \(M \equiv M' \in \text{ind}(\mathcal{K})\). Then \(M \not\forall V\) and \(M' \not\forall V\) where \(V, V'\) are 0-constructors and \(M \equiv V \equiv V' \in \text{ind}(\mathcal{K})\).

Central here is the validity condition on constraints for fcom and higher-dimensional intro terms. The assumption that \(\Xi\) is valid is a conservative approximation of the assumption that for every \(z\)-substitution \(\psi : \cdot \rightarrow \Psi\), there is some \(\xi \in \Xi\) with \(\xi \models \psi\). Thus, any dimension-fcom or higher intro term reduces, leaving as values only the 0-dimensional constructor terms. In non-empty dimension contexts, we still have a canonicity result, but in this case the value may be an fcom or higher intro.

### 4.3.2 Kan conditions

As planned, an \(\text{hcom} \in \text{ind}(\mathcal{K})\) steps to an fcom. The \(\text{hcom-\text{Kan}}\) conditions thus follow from Proposition 4.4, using Lemma 3.2 to mediate between \(\text{hcom}^{\pi, \emptyset}(\mathcal{K})\) and \(\text{fcom}\).

Coercion is more complicated. The operator \(\text{coe}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)}(\overline{V})\) is our first eager operator, that is, our first operator which evaluates its argument. For eager operators, properly defined, we can prove well-typedness by checking their behavior on values.

**Lemma 4.8.** Let \(A \equiv a \in \text{type}_{\pi, \emptyset}(\Psi)\) and \(a : A \not\exists B \equiv B \in \text{type}_{\pi, \emptyset}(\Psi)\), and let \(a + N, N^0 \equiv \Psi\) be eager. Suppose that for every \(\psi : \Psi' \rightarrow \Psi\) and \([\mathcal{A}_\psi](V, V')\), we have \(N\psi(V/a) \equiv N'\psi(V'/a) \in B\psi(V/a)\) \(\Psi'\). Then \(A \not\exists B \equiv N \equiv N' \in B \Psi\).

The behavior of \(\text{coe}\) on values is defined in Figure 9. Setting up the induction to prove the \(\text{coe-\text{Kan}}\) conditions is somewhat involved, but it all comes down to checking that \(\text{coe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)}(V)\) steps coherently to its reduc to any \(V\) val. Writing \(R\) for this reduc, we need that at any \(\psi\) for which \(V\psi \not\implies M\), we have \(\text{coe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)}(V) \equiv \text{coe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)}(V)\). If this is true and \(R\) itself is well-typed, then we can conclude \(\text{coe}^{\pi'-r'}_{z, \{\emptyset.\emptyset\}_K^{\pi'}(\overline{N}_j)}(V)\) is well-typed by Lemma 3.2.

When \(V\) is an fcom term, this is simple. On an fcom, \(\text{coe}^{\pi, \emptyset}(\mathcal{K})\) “pushes inside.” \(R\) is an fcom in which \(\text{coe}^{\pi, \emptyset}(\mathcal{K})\) is applied to the original tube and cap. If we take an aspect which makes the fcom argument reduce to a face, then the same aspect of \(R\) is equal to the coe of that face by Proposition 4.4(a) or (b).
With intro, more care is required. Naively, we might expect that coe pushes inside an intro term as well. However, this step is not coherent for higher constructors. To see why, let us consider the pushout \( \Pi := \Pi(A; B; C; F; G) \). Our naive proposal is to set
\[
\text{coe}^\succ_{z,A}(\text{glue}^x(P)) \mapsto \text{glue}^x(\text{coe}^\succ_{z,C}(P)).
\]
But consider what happens if we substitute 0 for \( x \). The left-hand side of this reduction then steps to \( \text{coe}^\succ_{z,A}(\text{left}(F(r/z)P)) \), which in turn presumably steps to \( \text{left}(\text{coe}^\succ_{z,A}(F(r/z)P)) \). The right-hand side is equal to \( \text{left}(F(r/z)(\text{coe}^\succ_{z,C}(P))) \). To have coherence, we need that \( \text{coe}^\succ_{z,A}(F(r/z)P) \) and \( F(r/z)(\text{coe}^\succ_{z,C}(P)) \) are equal in \( A(r/z) \). Unfortunately, coe does not commute with arbitrary functions.

Luckily, we can interpolate between the two with a line:
\[
\text{coe}^\succ_{z,A}(F(r/z)P) \xrightarrow{\text{coe}^\succ_{z,A}(F(\text{coe}^\succ_{z,C}(P)))} F(r/z)(\text{coe}^\succ_{z,C}(P)).
\]
We thus divide the operational semantics of coe on intro into two cases, one for 0-constructors and one for higher constructors. In the former case, we take the naive approach. For the latter case, we wrap the naive solution in an fcm which adjusts the relevant faces by a tube, pushing coe through the boundary functions. This corrects the coherence issue, and the proof goes through.

**Proposition 4.9.** \( \text{ind}(K) \text{type}_{\text{Kan}}[\Psi] \).

### 4.3.3 Elimination

For the elimination rule, the complexity is not so much in designing the operational semantics, but in specifying the data that must be supplied to an eliminator. To get an idea, let us consider again our examples. Suppose we want to define a dependent map from the pushout \( \Pi := \Pi(A; B; C; F; G) \) into some family \( h : \Pi \Rightarrow D \text{type}_{\text{Kan}}[\Psi] \). Intuitionally, we must provide maps

1. \( a : A \Rightarrow R_{\text{left}} D[\text{left}(a)/h] [\Psi] \),
2. \( b : B \Rightarrow R_{\text{right}} D[\text{right}(b)/h] [\Psi] \),
3. \( c : C \Rightarrow R_{\text{glue}} D[\text{glue}^x(c)/h] [\Psi, x] \),

which satisfy coherences

1. \( c : C \Rightarrow R_{\text{glue}} (1/x) \mapsto R_{\text{left}}[Fx/a] D[\text{glue}^0(c)/h] [\Psi] \),
2. \( c : C \Rightarrow R_{\text{glue}} (1/x) \mapsto R_{\text{right}}[Gx/b] D[\text{glue}^1(c)/h] [\Psi] \).

For the \((-1)\)-truncation \( |A| \), on the other hand, we should be able to construct a map into \( h : |A| \Rightarrow D \text{type}_{\text{Kan}}[\Psi] \) from maps

1. \( a : A \Rightarrow R_{\text{pt}} D[\text{pt}(a)/h] [\Psi] \),
2. \( \Gamma_{\text{path}} \Rightarrow R_{\text{path}} D[\text{path}^x(t_0/t_1)/h] [\Psi, x] \)

where \( \Gamma_{\text{path}} := (t_0 : |A|, t_1 : |A|, r_0 : D[t_0/h], r_1 : D[t_1/h]) \), satisfying

1. \( \Gamma_{\text{path}} \Rightarrow R_{\text{path}}(0/x) \mapsto r_0 D[\text{path}^0(t_0/t_1)/h] [\Psi] \),
2. \( \Gamma_{\text{path}} \Rightarrow R_{\text{path}}(1/x) \mapsto r_1 D[\text{path}^1(t_0/t_1)/h] [\Psi] \).

In the latter example, the terms \( r_0 \) and \( r_1 \) stand for the results of recursive calls at \( t_0 \) and \( t_1 \) respectively.

In general, the data required is an **elimination list** \( \mathcal{E} \) with a clause for each constructor:

\[
\mathcal{E} := \bullet \mid [\mathcal{E}, \ell : \mathcal{E}_1, \gamma, \delta, \rho, \mathcal{R}] \quad (\text{where } |\delta| = |\rho|)
\]

In each clause \( R \), we have access to variables \( \mathcal{E}_1 \) for the dimension parameters of the associated constructor, \( y \) and \( \delta \) for the non-recursive and recursive arguments, and \( \rho \) for the results of recursive calls on variables in \( \delta \). The judgment \( \mathcal{E} \equiv \mathcal{E}' : K \Rightarrow h.D[\Psi] \), defined in Figure 10, states that \( \mathcal{E} \) and \( \mathcal{E}' \) are equal lists covering some part of \( K \); we say \( \mathcal{E} \equiv \mathcal{E}' : K \Rightarrow h.D[\Psi] \) when \( \mathcal{E} \) and \( \mathcal{E}' \) cover all of \( K \).

To express the results of recursive calls and the coherence conditions, we define **dependent instantiations** of argument types and terms. For types, we set
\[
\{X\}_d(h.D.N) : = D[N/h]
\]
\[
\{(b:B) \rightarrow c\}_d(h.D.N) : = \{b\}_d(h.D; \text{app}(N, b))
\]
If \( h : A \Rightarrow D \text{type}_{\text{Kan}}[\Psi] \), then \( b : \{h\}_d(A) \Rightarrow \{h\}_d(h.D; b) \text{type}_{\text{Kan}}[\Psi] \). These types describe the results of calls to elim on recursive arguments, which are mediated by a dependent functorial action operator \( \text{functor} \). To express the coherence conditions on the clauses of an elimination list, we define dependent instantiation for terms in Figure 8. The term \( \{\theta.m\}_d(h.D.N; \mathcal{E}_1) \) combines the results \( \mathcal{E}_1 \) of recursive calls on terms \( \mathcal{E}_1 \) according to the shape of \( m \), producing the result of a recursive call on \( \{\theta.m\}_d(h.D; \mathcal{E}_1) \).

The operational semantics of \( \text{elim}_{h.D}(\cdot ; \mathcal{E}) \), which can be found in [13], is itself fairly simple. On intros, it steps to the appropriate clause of \( \mathcal{E} \), calling (the functorial action of) itself on the recursive arguments. On coms, it steps to a com in the target type, much as with coe. Each reduction rule is coherent in the target type, inducing a corresponding \( \beta \)-equation. From these we deduce a rule equating elims applied to equal arguments and elimination lists.

### 5 Homotopy fiber types

In this section, we pursue an inductive type which is not contained in our schema: the identity type. The identity type is an **indexed inductive type**, a family of types generated by constructors which introduce terms at specified indices (or fibers). The development of a schema for indexed inductives is beyond the scope of this paper, but...
Higher Inductive Types in Cubical Computational Type Theory

PL’18, January 01–03, 2018, New York, NY, USA

we lay the groundwork by implementing a representative example, the homotopy fiber type, which we use to define an identity type.

The fiber type, which was introduced to the first author by Steve Awodey, is the generic indexed inductive with a non-recursive constructor: given parameters $A, B$, and $a : A \Rightarrow F \in B$, the family $b : B \Rightarrow \text{fib}(A; B; a; F; b)$ is generated by a constructor $a : A \Rightarrow \text{refl}(a) \in \text{fib}(A; B; a; F; b)$. The type $B$ thus specifies the indexing type, $A$ the argument to the constructor $\text{refl}(a)$, and $F$ the index the term $\text{refl}(a)$ inherits. We can encode the identity type family $\text{Id}_A : a_0, a_1 : A \Rightarrow \text{Id}_A(a_0, a_1)$ as an instance of the fiber: $\text{Id}_A(M_0, M_1) := \text{fib}(A; A \times A; a_0, a_1; (M_0, M_1))$.

This corresponds to the homotopical interpretation of $\text{Id}_A$ as the factorization of the diagonal $\Delta : A \to A \times A$ [9]. The fiber type is one way of resolving a known issue in cubical type theories: the type $\text{Path}_A(M_0, M_1)$, which contains lines from $M_0$ to $M_1$, cannot be used as an identity type. While we can write equivalents of $\text{refl}$ and $\text{Path}_A$ for $\text{Path}_B$, they do not satisfy the expected $\beta$-rule exactly, only up to a higher path. The problem arises because it is impossible to detect when a line is reflexive (i.e., degenerate). A new type—here, the fiber type—is thus necessary to make CHTT a model of MLTT.

With CIs, we introduce formal fvs in order to account for composition of $n$-constructors. For indexed inductive types, a new issue arises: we need to account for the terms created by coercing from one fiber to another. For this, we introduce fcoes: formal (or free, or fiber) coercions.

Let $A, B$ type $\text{Kan} \left[ \Psi \right]$ and $a : A \Rightarrow F \in B \left[ \Psi \right]$ be given. We want to define a $\Psi$-relation $f(A, B, a; F)$ over $B$, a function taking every $\psi : \Psi \Rightarrow \Psi$ and $N \in B \Psi \left[ \Psi \right]$ to a $\Psi$-relation $f(A, B, a; F) \psi \left[ N \right]$. We define $f(A, B, a; F)$ to be the least $\Psi$-relation over $B$ such that

1. $\text{f}(A, B, a; F) \psi \left[ N \right] \Rightarrow \text{refl}(\text{M}(A), \text{refl}(\text{M}'))$ when
   - $M \Rightarrow M' \in A \psi \left[ \Psi '' \right]$,
   - $f \psi \left[ (\text{M}/a) \right] \Rightarrow N \psi \left[ \Psi '' \right] \in B \psi \left[ \Psi '' \right]$,
2. $f(A, B, a; F) \psi \left[ N \right] \Rightarrow (V, V')$ when
   - $\text{Fcom}(f(A, B, a; F) \psi \left[ N \right]) \Rightarrow (V, V')$,
3. $f(A, B, a; F) \psi \left[ N \right] \Rightarrow (\text{fcoe}_{x \Rightarrow \text{P}}(\text{O}), \text{fcoe}_{x \Rightarrow \text{P}}(\text{O}))$ when
   - $x \neq x'$,
   - $P \Rightarrow P' \in B \psi \left[ \Psi '' \right]$,
   - $P \Rightarrow (r/x) \Rightarrow N \psi \left[ \Psi '' \right] \in B \psi \left[ \Psi '' \right]$.

We require that $f(A, B, a; F) \psi \left[ N \right] \Rightarrow f(A, B, a; F) \psi \left[ N \right]$ for every $\psi'$ and that $f(A, B, a; F) \psi \left[ - \right]$ respects equality in $B$.

- $\text{Tm}(f(A, B, a; F) \psi \left[ P(r/x) \right]) \Rightarrow \text{O}'$.

Henceforth, we assume we are in a cubical type system where $\text{fib}(A; B; a; F; N)$ names $f(A, B, a; F) \left[ N \right]$. We will sometimes group the first three arguments to $\text{fib}$, writing $\text{fib}(A; B; a; F; N)$. The operational semantics for $\text{fib}$-related operators is given in Figure 11.

The pretype fib is thus composed of refl, fcoe, and fcoe terms. The term $\text{fcoe}_{x \Rightarrow \text{P}}(\text{O})$ coerces $O \in \text{fib}(A; B; a; F; P(r/x))$ along $P$ to obtain a term in $\text{fib}(A; B; a; F; P(r'/x))$. When $r = r'$, it steps to $O$; otherwise, it is a value.

We do not add all coercions as values, only those between fibers of the inductive. Thus, we still need to do some work to implement coercion in fib. To make up the difference, we introduce an eager operator $\text{coe}$ (total space coercion) which coeores along lines in the parameters $T := (A; B; a; F)$. The operational semantics of $\text{coe}$, which uses fcoe as well as coercion in $A$ and $B$, is defined in Figure 11. We can derive the following typing rules for $\text{coe}$:

**Proposition 5.1.** Let $\psi : (\Psi', x) \Rightarrow \Psi$. For any $N \in B \psi \left[ r/x \right] \left[ \Psi' \right]$ and $O \in \text{fib}(T \psi \left[ r/x \right]; N) \left[ \Psi' \right]$, we have

1. $\text{tcoe}^{x \Rightarrow \text{P}}(\text{O}) \in \text{fib}(T \psi \left[ r/x \right]; N) \left[ \Psi' \right]$,
2. $\text{tcoe}^{x \Rightarrow \text{P}}(\text{O}) \Rightarrow (\text{O}, O') \in \text{fib}(T \psi \left[ r/x \right]; N) \left[ \Psi' \right]$.

Note that $\text{coe}$ carries along the index $N$ via coercion in $B$. To implement coe in general, we use a $\text{coe}$ to get to the desired $A, B, a, F$, then follow up with an $\text{fcoe}$ to reach the correct index:

$fib(T; N)(r/x) \text{tcoe}^{x \Rightarrow \text{P}} (\text{O}) \text{fcoe}^{x \Rightarrow \text{P}} (N(r/x)) \Rightarrow fib(T; N)(r/x)$

The eliminator $\text{J}$ takes a clause $a; R$ for the refl case and satisfies the following typing rules:

**Proposition 5.2.** Let $b : B; h : \text{fib}(A; B; a; F; b) \Rightarrow D \text{type Kan} \left[ \Psi \right]$. If $a : A \Rightarrow R \in D \left[ F, \text{refl}(a)/b, h \right] \left[ \Psi \right]$, then

1. $b : B; h : \text{fib}(A; B; a; F; b) \Rightarrow \text{J}_{H, D, a}(h; a; R) \in D \left[ F, \text{refl}(a)/b, h \right] \left[ \Psi \right]$,
2. $a : A \Rightarrow \text{J}_{H, D, a}(\text{refl}(a)/a; R) \Rightarrow R \in D \left[ F, \text{refl}(a)/b, h \right] \left[ \Psi \right]$.

On fcoe and fcoe terms, J steps to a con or coe in the target type $D$, taking advantage of its Kan structure. On refl terms, J steps...
to the provided $R$. The second equation in Proposition 5.2 thus follows immediately from Lemma 3.2.

6 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 with canonicity properties.

In the first category, the earliest syntactic class is that of Sojakova’s $W$-quotients [27], 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 previous work on ordinary inductive types in HoTT [8]. More recently, Basold et al. [10] and Dybjær and Moenclaey [15] have given schemata for HITs which closely resemble ours, being described by grammars of argument types and terms, but which are limited to 1- and 2-constructors respectively. Altenkirch et al. [2] have defined a broad notion of quotient inductive-inductive type internally to type theory, but assume uniqueness of identity proofs—a principle inconsistent with univalence—to avoid coherence problems. Other work has focused on encoding more complex HITs in terms of simpler ones [20, 25, 31], but it is not clear that these techniques can account 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 an operational semantics and canonicity result.

In the second category, the main work is by Lumsdaine and Shulman [22], who define a notion of cell monad, a semantic specification of a HIT. Their schema includes all our examples and more, but has no obvious syntactic equivalent. Their level of generality also appears to be restrictive: they are not able to construct parameterized HIT in the same universe as their parameters, nor can their boundary terms mention composition. These issues arise from the ill-behavedness of fibrant replacement, a construction whose role is played by fcom and fceo in our setting. Our more fine-grained control over replacement seems to allow us to sidestep such issues.

Finally, we come to the cubical type theories. It is believed that higher inductive types are problematic for the BCH model because its dimension variables are linear rather than structural. However, the CCHM and CHTT theories each include a selection of higher inductives: both include the circle, the CCHM theory additionally supports general spheres and a $(−1)$-truncation type. The formal cartesian cubical type theory of Angiuli et al. [3] also supports pushouts. Huber [17] proves canonicity for CCHM, while CHTT satisfies a canonicity property by definition. Our contribution is to unify these disparate results in a general framework.

With regard to the fib type and its raison d’être—defining an Id-type in a cubical type theory—the original work is by Swan [28], who defined an Id-type for (a category equivalent to) the BCH model. Swan’s type has similarities to ours, being defined as a restriction of a fibrant replacement, but seems to use ld-specific details. Drawing inspiration from Swan, the CCHM theory includes an ld-type defined as a Path type with degeneracies labelled so that reflexive paths can be detected. Again, it is unclear whether this approach generalizes. Awodey [7] has given a cubical model in which Path and Id coincide, but which is not known to support a univalent universe.

In the future, we intend to extend the proof assistant RepPRL [29], which provides a mechanized proof theory for CHTT, with our schema for CITs. This will of course require greater attention to performance and to special cases where optimization is possible. There remains much room for generalization: besides the indexed CITs we hint at in Section 5, there is also a market for higher inductive-inductive types [30]. Even for ordinary CITs, Lumsdaine and Shulman [22] give an example which requires an extension allowing definition of argument terms by induction on positive types [13, §6.6].

Acknowledgments

We would like to thank Carlo Angiuli, Steve Awodey, Daniel Gratzer, Kuen-Bang Hou (Favonia), Dan Licata, Ed Morehouse, Anders Mörtberg, and Jonathan Sterling for valuable discussions and advice.

The authors gratefully acknowledge the support of the Air Force Office of Scientific Research through MURI grant FA9550-15-1-0053. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the AFOSR.

References