Higher Inductive Types in Cubical Computational Type Theory

EVAN CAVALLO, Carnegie Mellon University, USA
ROBERT HARPER, Carnegie Mellon University, USA

Homotopy type theory proposes *higher inductive types* (HITs) as a means of defining and reasoning about inductively-generated objects with higher-dimensional structure. As with the univalence axiom, however, homotopy type theory 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 variables. We extend the cartesian cubical computational type theory introduced by Angiuli et al. with a schema for indexed cubical inductive types (CITs), an adaptation of higher inductive types to the cubical setting. In doing so, we isolate the canonical values of a cubical inductive type and prove a canonicity theorem with respect to these values.

CCS Concepts: • **Theory of computation → Type theory**; **Operational semantics**;

Additional Key Words and Phrases: cubical type theory, higher inductive types, homotopy type theory

ACM Reference Format:
Evan Cavallo and Robert Harper. 2019. Higher Inductive Types in Cubical Computational Type Theory. *Proc. ACM Program. Lang.* 3, POPL, Article 1 (January 2019), 27 pages. https://doi.org/10.1145/3290314

1 INTRODUCTION

The basic premise of constructivism is that a sufficiently expressive programming language provides a foundation for all of mathematics. This principle is elegantly embodied in Martin Löf's extensional type theory [Martin-Löf 1982], which we will call *computational type theory* (CTT) following Nuprl terminology [Allen et al. 2006]. Beginning from an untyped language, Martin-Löf defines which program values represent propositions (i.e., types) and which values are canonical evidence for each proposition. A program is a proof of a proposition if it evaluates to a value which is canonical evidence for that proposition. In this setting, *proofs run*: a proof that a number exists is a program for computing that number.

While Martin-Löf's type theory is an expressive basis for formalizing mathematics, there is room for improvement in the area of *equality reasoning*. Specifically, the type equality relation is overly intensional: types can be equal only if they have exactly the same elements. In informal mathematics, on the contrary, it is common to treat sets as interchangeable even when they are merely isomorphic. The same issues (and others) arise in Martin-Löf's *intensional type theory* (ITT), a formal system which interprets into CTT but prioritizes proof-theoretic properties like decidability of equality judgments [Martin-Löf 1975]. Seeking to bridge this gap, Voevodsky proposed adding the *univalence axiom* to ITT, a rule which puts identifications between types A and B in correspondence with equivalences (roughly, isomorphisms) between A and B [Voevodsky 2010]. Here, *identifications*

Authors' addresses: Evan Cavallo, Computer Science Department, Carnegie Mellon University, USA, ecavallo@cs.cmu.edu; Robert Harper, Computer Science Department, Carnegie Mellon University, USA, rwh@cs.cmu.edu.

2475-1421/2019/1-ART1
https://doi.org/10.1145/3290314

between A and B are elements of the *identity type* $\mathsf{Id}_{\mathcal{U}}(A, B)$, which is characterized in ITT as the least reflexive binary relation on the universe of types $\mathcal{U}$. Crucial to this move is the consistency of univalence with that characterization, which is weak enough to permit multiple distinct elements of $\mathsf{Id}_{\mathcal{U}}(A, B)$ [Hofmann and Streicher 1998].

The standard computational interpretation of ITT into CTT, however, is incompatible with the univalence axiom. That interpretation takes the identity type $\mathsf{Id}_A(M, N)$ to the *equality type* $\mathsf{Eq}_A(M, N)$, which is inhabited by the single value $*$ if and only if M and N are equal as programs inhabiting A. The type $\mathsf{Eq}_{\mathcal{U}}(A, B)$ thus has at most one element, while there may be many equivalences between A and B. Moreover, it is essential to the interpretation that equality evidence is trivial. In ITT one can write a *coercion* function coe taking type identifications $\mathsf{Id}_{\mathcal{U}}(A, B)$ to functions $A \to B$. The image of coe in CTT runs its argument to the value $*$ and, thereby discovering that A and B are indeed equal, returns the identity function. If we add an element $\mathsf{ua}(E) \in \mathsf{Id}_{\mathcal{U}}(A, B)$ for every equivalence $E \in A \simeq B$, we need to explain how to run $\mathsf{coe}(\mathsf{ua}(E))$. Intuitively, $\mathsf{coe}(\mathsf{ua}(E))$ should reduce to the function underlying the equivalence E. But the problem is larger than this, for equality must also be a congruence: for any $F \in A \to B$, there is a function cong from $\mathsf{Id}_A(M, N)$ to $\mathsf{Id}_B(FM, FN)$. This means that an equivalence $E \in A \simeq B$ induces not only an element of $\mathsf{Id}_{\mathcal{U}}(A, B)$, but an element of $\mathsf{Id}_{\mathcal{U}}(A \to A, B \to B)$, an element of $\mathsf{Id}_{\mathcal{U}}(\mathsf{Id}_A(M, N), \mathsf{Id}_B(EM, EN))$, and so on. Introducing univalence adds a whole host of new elements to the identity types, and it is far from obvious how to implement coercion for each of them.

Enter *cubical type theory* [Angiuli et al. 2017b; Cohen et al. 2015], a theory of abstract coercions. Cubical type theory augments dependent type theory with a syntactic class of *dimension terms* $r, s, \ldots$, which can be either variables $x, y, z, \ldots$ or the constants $0, 1$. A term $M \in A$ which contains a dimension variable x is an abstract coercion, or *path*, between the *endpoints* $M\langle 0/x\rangle$ and $M\langle 1/x\rangle$ obtained by substituting a constant for x. A path of types—a type A containing a variable x—is actualized by coercion functions $\mathsf{coe}^{r \rightsquigarrow r'}_{x.A}$ which go from $A\langle r/x\rangle$ to $A\langle r'/x\rangle$ for each pair r, r'. Congruence, meanwhile, is immediate: if $M \in A$ is a path in x and $F \in A \to B$ is a function, then FM is a path in x with endpoints $FM\langle 0/x\rangle$ and $FM\langle 1/x\rangle$. The final ingredient is a second operation, hcom, which implements composition and inversion operations for paths. Paths can be internalized via *path types* $\mathsf{Path}_A(M, N)$ whose canonical elements are abstracted terms $\lambda^{\mathbb{I}}x.P$ where $P \in A$ is a path whose endpoints $P\langle 0/x\rangle$ and $P\langle 1/x\rangle$ are equal to M and N respectively. We can apply $P \in \mathsf{Path}_A(M, N)$ to a dimension r to get $P@r \in A$, with $(\lambda^{\mathbb{I}}x.P)@r$ reducing to $P\langle r/x\rangle$.

On the semantic side, the central change is that the meaning of a type A is defined by its canonical values in *each* context $\Psi = (x_1, \ldots, x_n)$ of dimension variables. In other words, we simultaneously specify the elements of A, the paths between elements of A, the paths between paths between elements of A, and so on. This gives us the freedom, for example, to specify that paths in $\mathcal{U}$ correspond to equivalences. Cohen et al. [2015] and Angiuli et al. [2018] have seen this through, defining cubical type theories which satisfy the univalence axiom when it is expressed in terms of paths. The former is a formal system with an accompanying canonicity proof [Huber 2016], while the latter is a computational type theory. (We discuss other differences in Section 5).

When types are specified by their values at each dimension, what is an inductively defined type? It is natural to imagine types generated not only by ordinary constructors but also *path constructors* connecting elements. Such types are known as *higher inductive types*. The univalence axiom and certain higher inductive types—specified axiomatically in terms of identity types—form the basis of *homotopy type theory*, an extension of ITT which has been used to formalize significant results from algebraic topology and homotopy theory [Univalent Foundations Program 2013]. We will use the term *cubical inductive type (CIT)* to refer to formulations in terms of path types, reserving *higher inductive type (HIT)* for formulations in terms of identity types.

$$
\begin{array}{l}
\mathsf{data}\ A/R \in \mathcal{U}\ \mathsf{where} \\
\quad \mathsf{in} \in A \to A/R \\
\quad \mathsf{rel}^x \in (a{:}A) \to (a'{:}A) \to R(a,a') \to A/R\ [x = 0 \hookrightarrow \mathsf{in}(a) \mid x = 1 \hookrightarrow \mathsf{in}(a')]
\end{array}
$$

Fig. 1. A homotopy quotient as a cubical inductive type.

$$
\begin{tikzcd}
\cdots \arrow[rr, bend left, "\mathsf{rel}^x(-3;-1;\ldots)"] & \mathsf{in}(-2) \arrow[rr, bend right, "\mathsf{rel}^x(-2;0;\ldots)"'] & \mathsf{in}(-1) \arrow[rr, bend left, "\mathsf{rel}^x(-1;1;\ldots)"] & \mathsf{in}(0) \arrow[rr, bend right, "\mathsf{rel}^x(0;2;\ldots)"'] & \mathsf{in}(1) \arrow[rr, bend left, "\mathsf{rel}^x(1;3;\ldots)"] & \mathsf{in}(2) & \cdots
\end{tikzcd}
$$

Fig. 2. $\mathbb{Z}$ mod 2 as the cubical inductive type $\mathbb{Z}/R_2$.

As a natural application, we can use cubical inductive types to take the quotient of a type A by a type-valued-relation $R \in A \to A \to \mathcal{U}$, a construction we will call the *homotopy quotient* A/R. We define A/R as generated by elements $\mathsf{in}(M)$ for each $M \in A$ and paths $\mathsf{rel}^x(M;N;P)$ for each $M, N \in A$ and $P \in R(M,N)$, with the prescription that $\mathsf{rel}^0(M;N;P)$ should reduce to $\mathsf{in}(M)$ and $\mathsf{rel}^1(M;N;P)$ to $\mathsf{in}(N)$. We introduce an informal notation for CIT definitions in Figure 1. The syntax $[x = 0 \hookrightarrow \cdots \mid x = 1 \hookrightarrow \cdots]$ specifies a constructor's reduction behavior when the specified equations on dimensions hold; we call these equations the *boundary* of the constructor. As an inductive type, A/R comes with an eliminator, which we can use to define functions $(q{:}A/R) \to C$ by case analysis. Much like an ordinary inductive type, it requires a function for each case—both element and path cases.

As an example, we could define the type of integers mod 2 as the homotopy quotient $\mathbb{Z}/R_2$ where $R_2(m,n) := \mathsf{Path}_{\mathbb{Z}}(m+2,n)$. Pictorially, this type appears as shown in Figure 2. Note, however, that the arrows shown are the *generating* paths: $\mathbb{Z}/R_2$ contains other paths, like one from $\mathsf{in}(2)$ to $\mathsf{in}(-2)$, which are obtained by composing or inverting generating paths. Using the eliminator for $\mathbb{Z}/R_2$, we can show it is equivalent to the type bool of booleans. Thanks to univalence, we can then coerce any result about $\mathbb{Z}/R_2$ to a result about bool and vice versa.

A consequence of introducing higher-dimensional structure is that we must be careful with definitions like these. For example, suppose we had instead tried to define $\mathbb{Z}$ mod 2 as $\mathbb{Z}/R'_2$ where $R'_2(m,n) := (k{:}\mathbb{Z}) \times \mathsf{Path}_{\mathbb{Z}}(m+2k,n)$. (We use the notation $(a{:}A) \times B$ and $(a{:}A) \to B$ for dependent pair and function types, respectively.) While $\mathbb{Z}/R_2$ contains one path from 0 to 2, $\mathbb{Z}/R'_2$ contains many: we can go straight 0 to 2, or from 0 to 4 to 2, or from 0 to 6 to -2 to 2, and so on. The result is that $\mathbb{Z}/R'_2$ is *not* equivalent to bool, because the former contains non-trivial loops like $0 \to 4 \to 2 \to 0$. In the case of $\mathbb{Z}$ mod 2, it is relatively simple to find a correct definition. However, this is not always the case. Luckily, we can obviate the issue with a more sophisticated CIT: the *0-truncation* $\|-\|_0$, which trivializes the higher structure of a type [Univalent Foundations Program 2013, §7.3].

To define the 0-truncation, we will first need a *circle*. The circle, defined in Figure 3, is one of the simplest cubical inductive types, generated by a point base and a loop at that point.

Intuitively, the 0-truncation $\|A\|_0$ of a type A is obtained by iteratively trivializing every loop—that is, every image of the circle—in A. The definition is shown in Figure 4. We begin with a constructor pt which includes A into $\|A\|_0$. Then, for every loop in $\|A\|_0$, i.e., every map $f : \mathbb{S}^1 \to \|A\|_0$, we recursively add a new element $\mathsf{hub}(f) \in \|A\|_0$. Finally, for each such f and element $s : \mathbb{S}^1$, we

$$
\begin{array}{l}
\mathsf{data}\ \mathsf{S}^1 \in \mathcal{U}\ \mathsf{where} \\
\quad \mathsf{base} \in \mathsf{S}^1 \\
\quad \mathsf{loop}^x \in \mathsf{S}^1\ [x = 0 \hookrightarrow \mathsf{base} \mid x = 1 \hookrightarrow \mathsf{base}]
\end{array}
$$

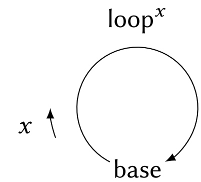

Fig. 3. The circle as a cubical inductive type.

$$
\begin{array}{l}
\mathsf{data}\ \|A\|_0 \in \mathcal{U}\ \mathsf{where} \\
\quad \mathsf{pt} \in A \to \|A\|_0 \\
\quad \mathsf{hub} \in (\mathsf{S}^1 \to \|A\|_0) \to \|A\|_0 \\
\quad \mathsf{spoke}^x \in (s{:}\mathsf{S}^1) \to (f{:}\mathsf{S}^1 \to \|A\|_0) \to \|A\|_0\ [x = 0 \hookrightarrow \mathsf{hub}(f) \mid x = 1 \hookrightarrow f s]
\end{array}
$$

Fig. 4. The 0-truncation of a type A.

$$
\begin{array}{l}
\mathsf{data}\ \mathsf{Id}_A \in A \to A \to \mathcal{U}\ \mathsf{where} \\
\quad \mathsf{refl} \in (a{:}A) \to \mathsf{Id}_A(a, a)
\end{array}
$$

Fig. 5. The identity type at A as an indexed inductive type.

connect $\mathsf{hub}(f)$ to the point $f s$ with a path $\mathsf{spoke}^x(s; f)$. The spoke paths connect each point on the loop in A to the hub, thereby trivializing the loop.

It is essential that this definition is recursive. We are trivializing loops by adding new elements, namely hubs and spokes, which may themselves give rise to new loops in $\|A\|_0$. By using a recursive constructor, however, we contract each new loop as it appears. Using the eliminators for $\mathbb{Z}/R_2'$ and $\|{-}\|_0$, we can show that $\|\mathbb{Z}/R_2'\|_0$ is equivalent to bool.

Even leaving the potential for path constructors aside, cubical type theory raises new questions for the computational interpretation of inductive types. In particular, problems arise when one considers *indexed inductive types*, families of types inductively generated by constructors at particular indices [Dybjer 1994]. The paradigmatic example is ITT's identity type, which is inductively generated by the reflexive identification refl as shown in Figure 5. To include the identity type in cubical type theory, we must define its coercion operator. In particular, given a path $P \in \mathsf{Path}_A(M, N)$, we can coerce the reflexive identification $\mathsf{refl}(M) \in \mathsf{Id}_A(M, M)$ to obtain a term $\mathsf{coe}^{0 \rightsquigarrow 1}_{x.\mathsf{Id}_A(M, P@x)}(\mathsf{refl}(M)) \in \mathsf{Id}_A(M, N)$. There are non-trivial paths, so there must be non-trivial identifications! Just as the generating paths of a cubical inductive type induce new paths by composition and inversion, the generating elements of an indexed inductive type induce new elements by coercion between indices.

The design of a schema for cubical inductive types thus presents a number of challenges. First, there is the design of the schema: what form can the arguments of a constructor take, and what form can its boundary take? How do we derive the elimination principle from an instance of the schema? Second, there are the computational questions: what are the canonical values of a higher inductive type? As we have seen, the necessity of supporting coe and hcom operations implies the existence of non-constructor values. What, then, are the values of a cubical inductive type? In what cases can we guarantee that a value is a constructor?

Contributions. We extend the cubical computational type theory of Angiuli et al. [2017b, 2018] with a schema for cubical inductive types which includes indexed types with n-dimensional recursive constructors. The language of argument types and boundary terms is specified by a small formal type theory. As a special case, we obtain an identity type for cubical type theory, making the theory a model of ITT. We obtain a canonicity theorem for this theory ensuring that every element of a cubical inductive type evaluates to a value. Such a value may be either a constructor or a derived term generated by coercion or composition. For non-indexed types, we can exclude the derived terms in the zero-dimensional case: any zero-dimensional element of a non-indexed cubical inductive type evaluates to a constructor for that type.

In Section 2, we recall the cubical computational type theory from Angiuli et al. [2018], defining the cubical programming language, typing judgments, and *Kan operations* coe and hcom. In Section 3, we introduce the features of our schema by way of example, considering the cases of the circle, 0-truncation, and identity type. We define their canonical values, then implement the Kan operations and eliminators for the types and prove their well-typedness. In Section 4, we define the general schema, implement the Kan operations for its instances, and formulate the elimination principle derived from an instance. We discuss related work in Section 5 and conclude with future work in Section 6. Complete proofs of our results can be found in our companion technical report [Cavallo and Harper 2018], which we henceforth cite as [TR].

ACKNOWLEDGMENTS

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

2 CUBICAL COMPUTATIONAL TYPE THEORY

In this section, we recall the framework of cubical computational type theory presented by Angiuli et al. [2018]. A computational type theory has two parts: an untyped programming language and a type system on that language which specifies types as partial equivalence relations (PERs) on the set of values. An element of a type is a term which evaluates to a value of that type. In the cubical setting, we add a sort of dimension terms to the language. The type system is then parameterized by a context of dimension variables. The elements of a type are those which evaluate to values of that type in a way which interacts coherently with dimension variable substitution. Finally, types are required to support the *Kan operations*, coe and hcom, which account for composition of and coercion along paths.

2.1 Cubical Programming Language

A cubical programming language consists of two sorts: dimensions r and terms M. The latter sort includes both terms which are "elements" and those which are "types;" these will be distinguished by the type system.

$$r ::= x \mid 0 \mid 1$$
$$M ::= (a{:}M) \to M \mid \lambda x.M \mid \mathsf{app}(M, M) \mid \mathsf{Path}_M(M, M) \mid \lambda^{\mathbb{I}}x.M \mid M@r \mid \cdots$$

We use either $\overrightarrow{M_i}$ or $\overline{M}$ to denote a list of terms. We impose a deterministic operational semantics on closed terms by way of judgments $M \longmapsto M'$ (M steps to M') and M val (M is a value). We use $\longmapsto^*$ for the transitive closure of $\longmapsto$, and write $M \Downarrow V$ (M evaluates to V) to mean that $M \longmapsto^* V$ with V val. While the operational semantics concerns terms without *term* variables, they can contain *dimension* variables. For example, Figure 6 shows the operational semantics of the circle constructors base and loop, which specifies that loop^x is a value path with endpoints which reduce

$$\frac{}{\mathsf{base}\ \mathsf{val}} \qquad \frac{}{\mathsf{loop}^x\ \mathsf{val}} \qquad \frac{\varepsilon \in \{0,1\}}{\mathsf{loop}^\varepsilon \longmapsto \mathsf{base}}$$

Fig. 6. Operational semantics of the circle constructors

to base. We write $M\ \mathsf{tm}\ [\Psi]$ to mean that $\mathrm{FD}(M) \subseteq \Psi$, $\mathrm{FD}(M)$ being the set of dimension variables free in M.

We write $\psi : \Psi' \to \Psi$ for a dimension variable substitution which takes $M\ \mathsf{tm}\ [\Psi]$ to $M\psi\ \mathsf{tm}\ [\Psi']$. We refer to these $M\psi$ as *cubical aspects* of M. The operational semantics need not be stable under substitution: $M \longmapsto M'$ does not imply $M\psi \longmapsto M'\psi$, nor does M val imply $M\psi$ val. For example, loop^x is a value, but $\mathsf{loop}^x\langle 0/x\rangle$ and $\mathsf{loop}^x\langle 1/x\rangle$ step.

We will start with the cubical language used by Angiuli et al. [2017c], which includes standard operators such as λ-abstraction and application, and extend it with new operators for inductive types as needed. The theorems we prove about it will hold for any language which extends the fragment we present. One could instead use Church-style encodings, though some extension to the λ-calculus is needed to accommodate dimensions.

2.2 Cubical Type Systems

Given an untyped programming language, we can define a type system over it in which types specify the evaluation behavior of programs. In Nuprl-style semantics, the denotation of a type A is a 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 canonical values in A.

In the cubical setting, the elements of a type A may contain dimension variables, so we stratify its denotation as a family of PERs $[\![A]\!]_\Psi$ indexed by dimension contexts $\Psi = (x_1, \ldots, x_n)$. We write $\varnothing$ for the empty context. At each Ψ, $[\![A]\!]_\Psi$ relates values $V\ \mathsf{tm}\ [\Psi]$, specifying the equal canonical values in that context. Actually, this is the special case where the term A itself is free of dimension variables. In general, the denotation of a type A in context Ψ is a family of PERs $[\![A]\!]_\psi$ indexed by substitutions $\psi : \Psi' \to \Psi$ into the context Ψ. For each $\psi : \Psi' \to \Psi$, $[\![A]\!]_\psi$ specifies the values of the aspect $A\psi$ in context Ψ'. We refer to such a substitution-indexed family of relations as a *Ψ-relation* (in this case, a *value Ψ-PER*). We write $\alpha, \beta, \ldots$ for Ψ-relations. We abbreviate $\alpha_\psi(M,M)$ as $\alpha_\psi(M)$ and $\alpha_{\mathrm{id}}(M,M')$ as $\alpha(M,M')$. For $\varnothing$-relations, we write α_Ψ instead of α_ψ, there being a unique substitution $\psi : \Psi \to \varnothing$. If α is a Ψ-relation and $\psi : \Psi' \to \Psi$, we write $\alpha\psi$ for the Ψ'-relation $(\alpha\psi)_{\psi'} := \alpha_{\psi\psi'}$.

As in the zero-dimensional case, a value Ψ-PER α extends by evaluation to a Ψ-PER on terms. Here, however, we don't want to include every term which evaluates to a value in α. Consider, for example, the following pathological term bad^r:

$$\frac{}{\mathsf{bad}^x \longmapsto \mathsf{loop}^x} \qquad \frac{\varepsilon \in \{0,1\}}{\mathsf{bad}^\varepsilon \longmapsto 17}$$

(This term can be encoded using the fhcom terms we introduce in Section 3.1.) Although bad^x evaluates to the canonical element loop^x of S^1, we do not want to include bad^x as an element of S^1, because its aspects $\mathsf{bad}^x\langle\varepsilon/x\rangle$ do not behave like elements of S^1. More insidiously, we may have a term which behaves as different elements of S^1 depending on the order in which we subject it to dimension substitutions and evaluation. To exclude such pathological terms, we introduce the following definition.

Definition 2.1. For any value Ψ-relation α, define a Ψ-relation $\textsc{Tm}(\alpha)$ by saying that $\textsc{Tm}(\alpha)_\psi(M, M')$ holds for $\psi : \Psi' \to \Psi$ when for every $\psi_1 : \Psi_1 \to \Psi'$ and $\psi_2 : \Psi_2 \to \Psi_1$, we have

$$\begin{array}{ccc} M\psi_1 \Downarrow V_1 & V_1\psi_2 \Downarrow V_2 & M\psi_1\psi_2 \Downarrow V_{12} \\ M'\psi_1 \Downarrow V_1' & V_1'\psi_2 \Downarrow V_2' & M'\psi_1\psi_2 \Downarrow V_{12}' \end{array}$$

for some $V_1, V_2, V_{12}, V_1', V_2', V_{12}'$ such that $\alpha_{\psi\psi_1\psi_2}(W, W')$ for every $W, W' \in \{V_2, V_2', V_{12}, V_{12}'\}$.

Intuitively, $\textsc{Tm}(\alpha)_\psi(M, M')$ holds when interleaving any pair of substitutions ψ_1, ψ_2 with evaluation of the terms M, M' in any order gives the same result up to α. When α is a PER, $\textsc{Tm}(\alpha)$ is a PER as well.

We will not work with this definition directly. Instead, we will present an interface of lemmas. First, note that if $\textsc{Tm}(\alpha)_\psi(M, M')$, then in particular M and M' evaluate to values related by α. Moreover, $\textsc{Tm}(\alpha)$ is always *stable*, where a Ψ-relation β is stable when $\beta_\psi(M, M')$ implies $\beta_{\psi\psi'}(M\psi', M'\psi')$ for all ψ', M, M'. The *value* Ψ-relations we use will typically not be stable, as valuehood itself is not stable. For example, we will define $[\![\mathsf{S}^1]\!]$ so that $[\![\mathsf{S}^1]\!]_\Psi(\mathsf{base}, \mathsf{base})$ for every Ψ and $[\![\mathsf{S}^1]\!]_{\Psi,x}(\mathsf{loop}^x, \mathsf{loop}^x)$ for every Ψ, x. However, we will not have $[\![\mathsf{S}^1]\!]_\Psi(\mathsf{loop}^0, \mathsf{loop}^0)$, as loop^0 is not even a value.

To prove that $\textsc{Tm}(\alpha)$ relates some pair of values, we make use of the following lemma.

Lemma 2.2 (Introduction [TR, A.2])**.** *Let α be a value Ψ-PER. If for every $\psi : \Psi' \to \Psi$, either $\alpha_\psi(M\psi, M'\psi)$ or $\textsc{Tm}(\alpha)_\psi(M\psi, M'\psi)$, then $\textsc{Tm}(\alpha)(M, M')$.*

In particular, if $\alpha_\psi(M\psi, M'\psi)$ for *every* ψ, then $\textsc{Tm}(\alpha)(M, M')$. Thus, for example, we will have $\textsc{Tm}([\![\mathsf{S}^1]\!])_\Psi(\mathsf{base}, \mathsf{base})$, because every aspect of base is itself base. We say α is *value-coherent* if it satisfies the stronger condition that $\alpha \subseteq \textsc{Tm}(\alpha)$. We will require this property of all types; it can fail, for example, if α contains loop^x but not base.

To prove reduction rules, we use the following analogue of a head expansion lemma.

Lemma 2.3 (Coherent expansion, [TR, A.3])**.** *Let α be a value Ψ-PER and let M, M' tm $[\Psi]$. If for every $\psi : \Psi' \to \Psi$, there exists M'' such that $M\psi \longmapsto^* M''$ and $\textsc{Tm}(\alpha)_\psi(M'', M'\psi)$, then $\textsc{Tm}(\alpha)(M, M')$.*

In particular, if $M\psi \longmapsto M'\psi$ for all ψ and $\textsc{Tm}(\alpha)(M')$ holds, then $\textsc{Tm}(\alpha)(M, M')$ holds. Thus, for example, we have $\textsc{Tm}([\![\mathsf{S}^1]\!])_\Psi(\mathsf{loop}^\varepsilon, \mathsf{base})$ for $\varepsilon \in \{0, 1\}$. As $\textsc{Tm}([\![\mathsf{S}^1]\!])$ is a PER, this implies $\textsc{Tm}([\![\mathsf{S}^1]\!])_\Psi(\mathsf{loop}^\varepsilon)$ for $\varepsilon \in \{0, 1\}$. Using Lemma 2.2, we can then prove $\textsc{Tm}([\![\mathsf{S}^1]\!])_\Psi(\mathsf{loop}^r)$ for any r. This is the typical style of argument for proving $\textsc{Tm}$ is closed under path constructors: the introduction rules for a constructor's boundary are used with coherent expansion to prove the introduction rule for the constructor itself.

A *candidate cubical type system* carves out a universe of Ψ-PERs and gives them syntactic names. Precisely, a candidate is a family $\tau = (\tau_\Psi)_\Psi$ of three-place relations $\tau_\Psi(A_0, B_0, \varphi)$ relating values A_0, B_0 tm $[\Psi]$ and (ordinary) PERs φ on values V, V' in context Ψ. As with $\textsc{Tm}$ in the case of value Ψ-PERs, a candidate induces relations $\textsc{PTy}(\tau)_\Psi(A, B, \alpha)$ on terms A, B tm $[\Psi]$ and value Ψ-PERs α. We will omit the definition, which is analogous to that of $\textsc{Tm}$, but note that $\textsc{PTy}(\tau)_\Psi(A, B, \alpha)$ implies $A \Downarrow A_0$ and $B \Downarrow B_0$ with $\tau_\Psi(A_0, B_0, \alpha_{\mathrm{id}})$ as well as $\textsc{PTy}(\tau)_{\Psi'}(A\psi, B\psi, \alpha\psi)$ for any $\psi : \Psi' \to \Psi$. A candidate τ is a *cubical type system* when a given A_0, B_0 are related by τ_Ψ to at most one φ, each $\tau_\Psi(-, -, \varphi)$ is a PER, and $\textsc{PTy}(\tau)_\Psi(A, B, \alpha)$ implies that α is value-coherent. We then write $[\![A]\!]$ for the unique α with $\textsc{PTy}(\tau)_\Psi(A, A, \alpha)$ when it exists. Angiuli et al. [2018] show how to construct a cumulative hierarchy of cubical type systems, each of which is closed under standard type formers and appears as a universe type in its successor.

Given a cubical type system τ, we can define type equality and element equality judgments.

$$\begin{aligned}
\tau \models A \doteq A' \ \mathsf{type_{pre}}\ [\Psi] \quad &:\Leftrightarrow \quad \exists\alpha.\ \mathrm{PT}\textsc{y}(\tau)_\Psi(A, A', \alpha),\\
\tau \models M \doteq M' \in A\ [\Psi] \quad &:\Leftrightarrow \quad \exists\alpha.\ \mathrm{PT}\textsc{y}(\tau)_\Psi(A, A', \alpha) \wedge \mathrm{T}\textsc{m}(\alpha)(M, M').
\end{aligned}$$

Note that each of these judgments is stable under dimension substitution. We will omit the prefix $\tau \models$ when the cubical type system is understood. We abbreviate $A \doteq A\ \mathsf{type_{pre}}\ [\Psi]$ as $A\ \mathsf{type_{pre}}\ [\Psi]$ and $M \doteq M \in A\ [\Psi]$ as $M \in A\ [\Psi]$. The open judgments are defined 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 $a{:}A \gg B \doteq B'\ \mathsf{type_{pre}}\ [\Psi]$ to hold when $B\psi[M/a] \doteq B'\psi[M'/a]\ \mathsf{type_{pre}}\ [\Psi']$ holds for every $\psi : \Psi' \to \Psi$ and $M \doteq M' \in A\psi\ [\Psi']$. We leave it to the reader to infer the definitions of context equality $\Gamma \doteq \Gamma'\ \mathsf{ctx_{pre}}\ [\Psi]$, equality in contexts $\overline{M} \doteq \overline{M}' \in \Gamma\ [\Psi]$, and the general open judgments $\Gamma \gg A \doteq A'\ \mathsf{type_{pre}}\ [\Psi]$ and $\Gamma \gg N \doteq N' \in A\ [\Psi]$. We will use the notation $\gamma : \Gamma$ to refer to the variables in a context Γ as a group.

Finally, we can use this notation to concisely state an elimination lemma for $\mathrm{T}\textsc{m}$. This lemma allows us to verify that an eager operator is well-typed on terms in $\mathrm{T}\textsc{m}(\alpha)$ by checking that it is well-typed on values in α.

Definition 2.4. We say that $a \vdash N\ \mathsf{tm}\ [\Psi]$ (a term N with one free variable a) is *eager* if for all $\psi : \Psi' \to \Psi$ and $M\ \mathsf{tm}\ [\Psi']$, we have $N\psi[M/a] \Downarrow W$ iff there exists $V\ \mathsf{tm}\ [\Psi']$ such that $M \Downarrow V$ and $N\psi[V/a] \Downarrow W$.

Lemma 2.5 (Elimination [TR, A.2])**.** *Let $A\ \mathsf{type_{pre}}\ [\Psi]$ and $a : A \gg B\ \mathsf{type_{pre}}\ [\Psi]$, and let $a \vdash N, N'\ \mathsf{tm}\ [\Psi]$ be eager. Suppose that for every $\psi : \Psi' \to \Psi$ and $[\![A]\!]_\psi(V, V')$, we have $N\psi[V/a] \doteq N'\psi[V'/a] \in B\psi[V/a]\ [\Psi']$. Then $a : A \gg N \doteq N' \in B\ [\Psi]$.*

2.3 Kan Types

We now have a cubical language and a notion of a pretype classifying values in this language. Finally, we distinguish *(Kan) types* as those pretypes which support the so-called Kan operations coe and hcom, which specify (a) the action of a path between types as a coercion, and (b) the means for composing paths to form other paths.

The first Kan operation is the *coercion* operation coe. The coe-*Kan conditions* require that for every r, r', and $M \in A\langle r/x\rangle\ [\Psi]$, we have $\mathsf{coe}^{r\rightsquigarrow r'}_{x.A}(M) \in A\langle r'/x\rangle\ [\Psi]$. Moreover, a degenerate coe must be trivial: $\mathsf{coe}^{r\rightsquigarrow r}_{x.A}(M) \doteq M \in A\langle r/x\rangle\ [\Psi]$. It is a consequence of these conditions that the map $\mathsf{coe}^{0\rightsquigarrow 1}_{x.A}$ is an equivalence with inverse $\mathsf{coe}^{1\rightsquigarrow 0}_{x.A}$. To give an idea of why this is true, observe that the path shown below connects any input $M \in A\langle 0/x\rangle\ [\Psi]$ to the result $\mathsf{coe}^{1\rightsquigarrow 0}_{x.A}(\mathsf{coe}^{0\rightsquigarrow 1}_{x.A}(M))$ of one round trip; a similar path works for the other round trip.

$$\begin{array}{ccc}
M & \xrightarrow{\quad \mathsf{coe}^{y\rightsquigarrow 0}_{x.A}(\mathsf{coe}^{0\rightsquigarrow y}_{x.A}(M)) \quad} & \mathsf{coe}^{1\rightsquigarrow 0}_{x.A}(\mathsf{coe}^{0\rightsquigarrow 1}_{x.A}(M)) \\
0 & \xrightarrow[\quad y \quad]{} & 1
\end{array}$$

This means that every path $P \in \mathsf{Path}_{\mathcal{U}}(A, B)\ [\Psi]$ gives rise to an equivalence between its endpoints A and B. The univalence axiom asserts that this path-to-equivalence operation is itself an equivalence; its computational interpretation is given in Angiuli et al. [2018], but we will not need it here.

The second Kan operation is the *homogeneous composition* operation hcom. In its most general form, hcom takes a term called a *cap* and adjusts certain aspects of that term along a set of lines called a *tube*. The general case encompasses the many ways of composing and inverting of n-dimensional cubes and moreover enables us to prove properties of these operations, such as associativity of composition up to a path.

To state the hcom-Kan conditions, we first have to introduce *constraints*, which are equations $r = r'$ on dimension terms. We write ξ for constraints and Ξ or $\overrightarrow{\xi_i}$ for lists of constraints. We write $\models \Xi$ to mean that every constraint in Ξ is true, i.e, that each is of the form $r = r$ for some r. We define restricted judgments $A \doteq A'\ \mathsf{type_{pre}}\ [\Psi \mid \Xi]$ and $M \doteq M' \in A\ [\Psi \mid \Xi]$ expressing that an equation holds on the part of the context Ψ where the constraints are true:

$$\begin{array}{rcl} A \doteq A'\ \mathsf{type_{pre}}\ [\Psi \mid \Xi] & :\Leftrightarrow & \forall \psi : \Psi' \to \Psi.\ (\models \Xi\psi \to A\psi \doteq A'\psi\ \mathsf{type_{pre}}\ [\Psi']) \\ M \doteq M' \in A\ [\Psi \mid \Xi] & :\Leftrightarrow & \forall \psi : \Psi' \to \Psi.\ (\models \Xi\psi \to M\psi \doteq M'\psi \in A\psi\ [\Psi']) \end{array}$$

A *tube* is a list $(\xi_1 \hookrightarrow y.N_1,\ \ldots,\ \xi_n \hookrightarrow y.N_n)$ of constraints ξ_i paired with lines $y.N_i$ (terms in an abstracted dimension y). A tube $\overrightarrow{\xi_i \hookrightarrow y.N_i}$ is well-typed in A at Ψ when $N_i \doteq N_j \in A\ [\Psi, y \mid \xi_i, \xi_j]$ holds for each i, j—in particular, $N_i \in A\ [\Psi, y \mid \xi_i]$ for each i. These equations ensure that the terms N_i agree where the constraints overlap, so that a tube is a "partial cube" defined on their union. Finally, we impose a *validity restriction* on the shape of tubes. We say a constraint list Ξ is *valid* when there is some r such that either $(r = r) \in \overrightarrow{\xi_i}$ or both $(r = 0) \in \Xi$ and $(r = 1) \in \Xi$. We require tubes to have valid constraint sets, which will allow us to prove a stronger canonicity theorem in Section 4.4.

We say that a term $M \in A\ [\Psi]$ is a *cap for* $\overrightarrow{\xi_i \hookrightarrow y.N_i}$ *at* r when $M \doteq N_i\langle r/y\rangle \in A\ [\Psi \mid \xi_i]$ for each i. Intuitively, this means M fits into the tube at position r. As an example, the term loop^x is a cap for the tube $(x = 0 \hookrightarrow y.\mathsf{base}, x = 1 \hookrightarrow y.\mathsf{loop}^y)$ at 0:

$$\begin{tikzcd} \cdot \arrow[r, "\mathsf{loop}^x"] \arrow[d, "\mathsf{base}"'] & \cdot \arrow[d, "\mathsf{loop}^y"] \\ \cdot & \cdot \end{tikzcd}$$

The hcom operator takes a valid tube $\overrightarrow{\xi_i \hookrightarrow y.N_i}$ and a cap M at a given r and produces a cap at any r'. Syntactically, the hcom-*Kan conditions* require that $\mathsf{hcom}_A^{r \rightsquigarrow r'}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}) \in A\ [\Psi]$ with $\mathsf{hcom}_A^{r \rightsquigarrow r'}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}) \doteq N_i\langle r'/y\rangle \in A\ [\Psi \mid \xi_i]$ for all i. As with coe, we also require that degenerate hcoms trivialize: $\mathsf{hcom}_A^{r \rightsquigarrow r}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}) \doteq M \in A\ [\Psi]$. Thus, in our previous example, we have the following composite:

$$\begin{tikzcd} \cdot \arrow[r, "\mathsf{loop}^x"] \arrow[d, "\mathsf{base}"'] \arrow[dr, phantom, "\mathsf{hcom}_A^{0 \rightsquigarrow y}(\mathsf{loop}^x; \cdots)"] & \cdot \arrow[d, "\mathsf{loop}^y"] \\ \cdot \arrow[r, dashed, "\mathsf{hcom}_A^{0 \rightsquigarrow 1}(\mathsf{loop}^x; \cdots)"'] & \cdot \end{tikzcd}$$

The term in the center is a square whose boundary is given by the surrounding paths. In this example, the term $\mathsf{hcom}_A^{0 \rightsquigarrow 1}(\mathsf{loop}^x; \cdots)$ is the result of composing the paths loop^x and loop^y together. In general, a two-dimensional $\mathsf{hcom}_A^{0 \rightsquigarrow 1}$ like this one composes (1) the inverse of the left face, (2) the top face, and (3) the right face into a single path. In this case, the left face is degenerate, so we obtain the ordinary composition of two paths. By leaving the top and right face degenerate, we could instead obtain the inverse of the left face.

$$\frac{(\forall i)\ \not\models \xi_i \qquad r \neq r'}{\mathsf{fhcom}^{r \rightsquigarrow r'}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})\ \mathsf{val}} \qquad \frac{(\forall i)\ \not\models \xi_i \qquad r = r'}{\mathsf{fhcom}^{r \rightsquigarrow r'}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}) \longmapsto M}$$

$$\frac{\models \xi_i \qquad (\forall j < i)\ \not\models \xi_j}{\mathsf{fhcom}^{r \rightsquigarrow r'}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}) \longmapsto N_i\langle r'/y\rangle}$$

Fig. 7. Operational semantics of fhcom.

Equal pretypes $A \doteq A'$ $\mathsf{type_{pre}}$ $[\Psi]$ are *equally Kan*, written $A \doteq A'$ $\mathsf{type_{Kan}}$ $[\Psi]$, when each of their aspects $A\psi \doteq A'\psi$ $\mathsf{type_{pre}}$ $[\Psi']$ satisfy the Kan conditions with equal implementations of hcom and coe.

3 CUBICAL INDUCTIVE TYPES BY EXAMPLE

We will work our way up to the general case with a series of examples, each of which introduces new components of the machine. In Section 3.1, we define the circle and implement its Kan operations and eliminator. The circle is simplistic in a few ways: it takes no parameters or indices and its constructors are non-recursive. In Section 3.2, we upgrade to the 0-truncation, which takes a type parameter (but not an index) and has recursive constructors. In Section 3.3, we cover the identity type, which has no higher constructors but is indexed.

3.1 Circle

As shown in Figure 3, the circle S^1 is generated by a point base and a path loop connecting that point to itself. We define the operational semantics of the constructors as in Figure 6. For S^1 to be Kan, however, it must contain other values: we can, for example, invert loop or compose it with itself any number of times to obtain new paths. In this case, we do not have to worry about satisfying the coe-Kan conditions: as S^1 contains no dimension variables, we can define $\mathsf{coe}^{r \rightsquigarrow r'}_{x.\mathsf{S}^1}$ to be the identity function. However, we do need to add values to account for hcom. For this purpose, we introduce the fhcom operator, the *free* or *formal* implementation of hcom.

As shown in Figure 7, fhcom takes the same arguments as hcom with the exception of the type subscript. When one of the tube constraints ξ_i holds, $\mathsf{fhcom}^{r \rightsquigarrow r'}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})$ steps to the corresponding tube face $N_i\langle r'/y\rangle$; when $r = r'$, it steps to the cap M. Otherwise, it is simply a value. We can use fhcom to implement hcom for any type at the cost of adding new values to its PER. We define a monotone operator $\textsc{Fhcom}$ on Ψ-relations which takes α to the Ψ-relation of fhcom values built from arguments in $\textsc{Tm}(\alpha)$:

$$\textsc{Fhcom}(\alpha)_\psi := \{(\mathsf{fhcom}^{r \rightsquigarrow r'}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}), \mathsf{fhcom}^{r \rightsquigarrow r'}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i'})) \mid \textsc{Tm}(\alpha)_\psi(M, M') \wedge \cdots\},$$

where the omitted conditions prescribe that $\overrightarrow{\xi_i \hookrightarrow y.N_i}$ and $\overrightarrow{\xi_i \hookrightarrow y.N_i'}$ are equal tubes in $\textsc{Tm}(\alpha)$ with equal caps M and M' at r. As with the corresponding Kan condition, we require the list $\overrightarrow{\xi_i}$ of tube constraints in an fhcom to be valid. We define the circle as a least fixed-point of $\varnothing$-relations: $[\![\mathsf{S}^1]\!] := \mu\alpha.\textsc{Circle}(\alpha)$ where $\textsc{Circle}$ is the monotone operator

$$\textsc{Circle}(\alpha)_\Psi := \{(\mathsf{base}, \mathsf{base})\} \cup \{(\mathsf{loop}^x, \mathsf{loop}^x) \mid x \in \Psi\} \cup \textsc{Fhcom}(\alpha)_\Psi.$$

Using Lemmas 2.2 and 2.3, we can show that $\textsc{Tm}([\![\mathsf{S}^1]\!])$ contains base and loop^r and is closed under fhcom terms, which implies in particular that $[\![\mathsf{S}^1]\!]$ is value-coherent. At this point, it is trivial to implement $\mathsf{coe}_{x.\mathbb{S}^1}$ and $\mathsf{hcom}_{\mathbb{S}^1}$:

$$\frac{}{\mathsf{coe}^{r\rightsquigarrow r'}_{x.\mathbb{S}^1}(M) \longmapsto M} \qquad \frac{}{\mathsf{hcom}^{r\rightsquigarrow r'}_{\mathbb{S}^1}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}) \longmapsto \mathsf{fhcom}^{r\rightsquigarrow r'}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})}$$

As we have added elements to $\mathbb{S}^1$ besides the generators, we may wonder if the type we defined is really a circle: can we implement an elimination rule? We should be able to define a function from the circle into a type family C by providing the base and loop cases, per the rule below.

$$\frac{\begin{array}{c} M \in \mathbb{S}^1\ [\Psi] \qquad N \in C[\mathsf{base}/a]\ [\Psi] \\ P \in C[\mathsf{loop}^z/a]\ [\Psi, z] \qquad (\forall \varepsilon \in \{0,1\})\ P \doteq N \in C[\mathsf{loop}^z/a]\ [\Psi \mid z = \varepsilon] \end{array}}{\mathbb{S}^1\text{-}\mathsf{elim}_{a.C}(M; N, z.P) \in C[M/a]\ [\Psi]}$$

The eliminator $\mathbb{S}^1\text{-}\mathsf{elim}_{a.C}(M; N, z.P)$ shown above takes an element of the circle, a term N providing the output for the case $M = \mathsf{base}$, and a term $z.P$ in one dimension variable providing the output for the case $M = \mathsf{loop}^z$. Note the final "coherence" premise of this rule, which ensures that the endpoints of the loop case $z.P$ line up with the base case N. It is clear how $\mathbb{S}^1\text{-}\mathsf{elim}$ ought to step applied to base and loop terms:

$$\frac{}{\mathbb{S}^1\text{-}\mathsf{elim}_{a.C}(\mathsf{base}; N, z.P) \longmapsto M} \qquad \frac{}{\mathbb{S}^1\text{-}\mathsf{elim}_{a.C}(\mathsf{loop}^x; N, z.P) \longmapsto P\langle x/z\rangle}$$

From Lemma 2.3, we can see immediately that $\mathbb{S}^1\text{-}\mathsf{elim}_{a.C}(\mathsf{base}; N, z.P) \doteq M \in C[\mathsf{base}/a]\ [\Psi]$. To prove the corresponding rule for loop,

$$\mathbb{S}^1\text{-}\mathsf{elim}_{a.C}(\mathsf{loop}^r; N, z.P) \doteq P\langle r/z\rangle \in C[\mathsf{loop}^r/a]\ [\Psi],$$

we make essential use of the "coherence" premise. To apply Lemma 2.3, we need to know that $\mathbb{S}^1\text{-}\mathsf{elim}_{a.C}(\mathsf{loop}^r; N, z.P)\psi$ steps to something equal to $P\langle r/z\rangle\psi$ for any substitution ψ. When $r\psi$ is a variable, this is immediate. When $r\psi \in \{0,1\}$, we have

$$\mathbb{S}^1\text{-}\mathsf{elim}_{a.C}(\mathsf{loop}^r; N, z.P)\psi \longmapsto \mathbb{S}^1\text{-}\mathsf{elim}_{a.C}(\mathsf{base}; N, z.P)\psi \longmapsto N\psi,$$

a reduct which is equal to $P\langle r/z\rangle\psi$ by the "coherence" premise.

Finally, we have to define the behavior of $\mathbb{S}^1\text{-}\mathsf{elim}$ on fhcom terms. For this, we can take advantage of the fact that C is a family of *types* and so has Kan structure of its own. Essentially, we want to map fhcoms in $\mathbb{S}^1$ to hcoms in C. Because we are defining a *dependent* map, the solution is a bit more complex than this: we need to define *heterogeneous composition*, or com [Angiuli et al. 2017c, Theorem 44].

Lemma 3.1 (Heterogeneous composition). *Define*

$$\mathsf{com}^{r\rightsquigarrow r'}_{y.A}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}) := \mathsf{hcom}^{r\rightsquigarrow r'}_{A\langle r'/y\rangle}(\mathsf{coe}^{r\rightsquigarrow r'}_{y.A}(M); \overrightarrow{\xi_i \hookrightarrow y.\mathsf{coe}^{y\rightsquigarrow r'}_{y.A}(N_i)}).$$

For A $\mathsf{type}_{\mathsf{Kan}}$ $[\Psi, y]$, this definition satisfies the following rules:

$$\frac{M \in A\langle r/y\rangle\ [\Psi] \qquad (\forall i,j)\ N_i \doteq N_j \in A\ [\Psi, y \mid \xi_i] \qquad (\forall i)\ N_i\langle r'/y\rangle \doteq M \in A\langle r/y\rangle\ [\Psi \mid \xi_i, \xi_j]}{\begin{array}{c} \mathsf{com}^{r\rightsquigarrow r'}_{y.A}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}) \in A\langle r'/y\rangle\ [\Psi] \\ \mathsf{com}^{r\rightsquigarrow r'}_{y.A}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}) \doteq M \in A\langle r'/y\rangle\ [\Psi \mid r = r'] \\ \mathsf{com}^{r\rightsquigarrow r'}_{y.A}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}) \doteq N_i\langle r'/y\rangle \in A\langle r'/y\rangle\ [\Psi \mid \xi_i] \end{array}}$$

Heterogeneous composition combines the functions of coe and hcom: it carries the cap M across the type line $y.A$ while simultaneously adjusting by tube faces $\xi_i \hookrightarrow y.N_i$ which extend along the

$$\frac{}{\mathsf{pt}(M)\ \mathsf{val}} \qquad \frac{}{\mathsf{hub}(F)\ \mathsf{val}}$$

$$\frac{}{\mathsf{spoke}^x(F;M)\ \mathsf{val}} \qquad \frac{}{\mathsf{spoke}^0(F;M) \longmapsto \mathsf{hub}(F)} \qquad \frac{}{\mathsf{spoke}^1(F;M) \longmapsto FM}$$

Fig. 8. Operational semantics of the 0-truncation constructors

type line. With this definition in hand, we can specify the operational semantics of S^1-elim on an fhcom term.

$$\frac{r \neq r' \qquad (\forall i)\ \not\models \xi_i \qquad F^y := \mathsf{fhcom}^{r \rightsquigarrow y}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})}{\begin{array}{c}\mathsf{S}^1\text{-}\mathsf{elim}_{a.C}(\mathsf{fhcom}^{r \rightsquigarrow r'}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}); N, z.P) \longmapsto \\ \mathsf{com}^{r \rightsquigarrow r'}_{y.C[F^y/a]}(\mathsf{S}^1\text{-}\mathsf{elim}_{a.C}(M; N, z.P); \overrightarrow{\xi_i \hookrightarrow y.\mathsf{S}^1\text{-}\mathsf{elim}_{a.C}(N_i; N, z.P)})\end{array}}$$

Let us check that this definition makes type sense. The recursive call $\mathsf{S}^1\text{-}\mathsf{elim}_{a.C}(M; N, z.P)$ lands in $C[M/a]$, which is equal to $C[F^y/a]\langle r/y\rangle$ because F^r is equal to M. Likewise, $\mathsf{S}^1\text{-}\mathsf{elim}_{a.C}(N_i; N, z.P)$ lands in $C[N_i/a]$, which is equal to $C[F^y/a]$ under ξ_i because F^y is equal to N_i under ξ_i. By Lemma 3.1, composing these terms gives a result in $C[F^y/a]\langle r'/y\rangle$, which is precisely the desired type $C[\mathsf{fhcom}^{r \rightsquigarrow r'}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})/a]$.

Note that the motive annotation $a.C$ on the eliminator is not present for type-checking purposes, but rather is essential to the operational semantics: the eliminator defers to the motive to handle fhcom values.

3.2 0-Truncation

The 0-truncation, shown in Figure 4, is a *parameterized* inductive type: we want to construct $\|A\|_0\ \mathsf{type}_{\mathsf{Kan}}\ [\Psi]$ for every $A\ \mathsf{type}_{\mathsf{Kan}}\ [\Psi]$. Unlike S^1, the term $\|A\|_0$ can contain dimension variables if A does, so we can no longer get away with a degenerate definition of coe.

Besides coercion, the definition of truncation closely follows that of the circle. We define the operational semantics of the 0-truncation constructors in Figure 8: a constructor steps when a boundary constraint holds and is a value otherwise. The Ψ-PER $[\![\|A\|_0]\!]$ is defined as the least closed under constructor values and fhcom values, and we can show that $\textsc{Tm}([\![\|A\|_0]\!])$ validates the expected introduction rules. Once again, we implement hcom via fhcom.

Now, we come to coe. For every line $z.\|A\|_0$ and endpoints r and r', we must define the function $\mathsf{coe}^{r \rightsquigarrow r'}_{z.\|A\|_0}$ from $\|A\|_0\langle r/z\rangle$ to $\|A\|_0\langle r'/z\rangle$. We will do this by evaluating the input element of $\|A\|_0\langle r/z\rangle$ and analyzing the structure of its value, using the Kan structures on S^1 and A at the leaves to coerce. (For illustrative purposes, we will forget for the moment that coercion in S^1 is the identity function.) First, we define coe on pt terms in the natural way, using the Kan structure on A:

$$\frac{}{\mathsf{coe}^{r \rightsquigarrow r'}_{z.\|A\|_0}(\mathsf{pt}(M)) \longmapsto \mathsf{pt}(\mathsf{coe}^{r \rightsquigarrow r'}_{z.A}(M))}$$

Given $M \in A\langle r/z\rangle\ [\Psi]$, we have $\mathsf{coe}^{r \rightsquigarrow r'}_{z.A}(M) \in A\langle r'/z\rangle\ [\Psi]$, so we can apply pt to get a term in $\|A\|_0\langle r'/z\rangle$. For hub, we similarly push the coe into the argument:

$$\frac{}{\mathsf{coe}^{r \rightsquigarrow r'}_{z.\|A\|_0}(\mathsf{hub}(F)) \longmapsto \mathsf{hub}(\mathsf{coe}^{r \rightsquigarrow r'}_{z.\mathsf{S}^1 \to \|A\|_0}(F))}$$

In this case, we reduce to a coe in the compound type $\mathsf{S}^1 \to \|A\|_0$, which is itself defined in terms of coe in S^1 and $\|A\|_0$. Coercion for function types is defined in [Angiuli et al. 2018, §5.3]; for our purposes, we do not need to know the definition.

Finally, we have the path constructor spoke^x. The rub will be to define the reduction for $\mathsf{coe}^{r\rightsquigarrow r'}_{z.\|A\|_0}(\mathsf{spoke}^x(M;F))$ in a coherent way: when we try to show it is well-typed with Lemma 2.3, we will need its reduct's $x = 0$ face to agree with $\mathsf{coe}^{r\rightsquigarrow r'}_{z.\|A\|_0}(\mathsf{hub}(F))$ and its $x = 1$ face to agree with $\mathsf{coe}^{r\rightsquigarrow r'}_{z.\|A\|_0}(FM)$. As a first cut, we might try to define

$$\mathsf{coe}^{r\rightsquigarrow r'}_{z.\|A\|_0}(\mathsf{spoke}^x(M;F)) \overset{?}{\longmapsto} \mathsf{spoke}^x(\mathsf{coe}^{r\rightsquigarrow r'}_{z.\mathsf{S}^1}(M); \mathsf{coe}^{r\rightsquigarrow r'}_{z.\mathsf{S}^1\to\|A\|_0}(F))$$

However, consider the $x = 1$ face of this definition. If we substitute 1 for x on the left hand side, we obtain a term which reduces to $\mathsf{coe}^{r\rightsquigarrow r'}_{z.\|A\|_0}(FM)$. If, on the other hand, we substitute 1 for x on the right hand side, we reduce to $\mathsf{coe}^{r\rightsquigarrow r'}_{z.\mathsf{S}^1\to\|A\|_0}(F)(\mathsf{coe}^{r\rightsquigarrow r'}_{z.\mathsf{S}^1}(M))$. These are not necessarily equal; coe does not commute with function application in general.

Luckily, coe *does* commute with all operators *up to a path*. Consider the case of an arbitrary function $G \in B \to C$. Given $N \in B$, we can construct a y-line between $\mathsf{coe}^{r\rightsquigarrow r'}_{z.C}(GN)$ and $G(\mathsf{coe}^{r\rightsquigarrow r'}_{z.B}(N))$ like so:

$$\begin{array}{ccc} \mathsf{coe}^{r\rightsquigarrow r'}_{z.C}(GN) & \xrightarrow{\quad \mathsf{coe}^{y\rightsquigarrow r'}_{z.C}(G(\mathsf{coe}^{r\rightsquigarrow y}_{z.B}(N))) \quad} & G(\mathsf{coe}^{r\rightsquigarrow r'}_{z.C}(N)) \\ r & \xrightarrow{\qquad\qquad\quad y \qquad\qquad\quad} & r' \end{array}$$

When $y = r$, the inner coe in the term $\mathsf{coe}^{y\rightsquigarrow r'}_{z.C}(G(\mathsf{coe}^{r\rightsquigarrow y}_{z.B}(N)))$ simplifies and we are left with $\mathsf{coe}^{r\rightsquigarrow r'}_{z.C}(GN)$; when $y = r'$, the outer coe simplifies and we have $G(\mathsf{coe}^{r\rightsquigarrow r'}_{z.C}(N))$. We can think of the interpolating term as coercing N from r to an intermediate point y, applying the function G, and then coercing GN the rest of the way from y to r'.

Instantiating this argument with function application, we see that our attempted definition has the correct boundary up to a path but not up to equality. This is precisely the use case for hcom: we want to adjust the boundary of a term by a tube. Using an fhcom (i.e., an hcom in $\|A\|_0$), we thus define

$$\frac{\overline{T} := \left[\begin{array}{l} x = 0 \hookrightarrow y.\mathsf{coe}^{r\rightsquigarrow r'}_{z.\|A\|_0}(\mathsf{hub}(F)), \\ x = 1 \hookrightarrow y.\mathsf{coe}^{y\rightsquigarrow r'}_{z.\|A\|_0}(\mathsf{coe}^{r\rightsquigarrow y}_{z.\mathsf{S}^1\to\|A\|_0}(F)(\mathsf{coe}^{r\rightsquigarrow y}_{z.\mathsf{S}^1}(M))) \end{array}\right]}{\mathsf{coe}^{r\rightsquigarrow r'}_{z.\|A\|_0}(\mathsf{spoke}^x(M;F)) \longmapsto \mathsf{fhcom}^{r'\rightsquigarrow r}(\mathsf{spoke}^x(\mathsf{coe}^{r\rightsquigarrow r'}_{z.\mathsf{S}^1}(M); \mathsf{coe}^{r\rightsquigarrow r'}_{z.\mathsf{S}^1\to\|A\|_0}(F)); \overline{T})}$$

The $x = 0$ face is a constant line, since no boundary correction is necessary there, while the $x = 1$ face slides the coercion out of the function application.

Finally, we have to handle the case of fhcom values. Here, we can simply push the coe inside:

$$\frac{}{\mathsf{coe}^{r\rightsquigarrow r'}_{z.\|A\|_0}(\mathsf{fhcom}^{s\rightsquigarrow s'}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})) \longmapsto \mathsf{fhcom}^{s\rightsquigarrow s'}(\mathsf{coe}^{r\rightsquigarrow r'}_{z.\|A\|_0}(M); \overrightarrow{\xi_i \hookrightarrow y.\mathsf{coe}^{r\rightsquigarrow r'}_{z.\|A\|_0}(N_i)})}$$

It is straightforward to check that this definition has all the right faces. This completes the definition of coe in $\|A\|_0$.

Our final task is to define the eliminator for $\|A\|_0$. We will follow the same template as for the circle, but now we must account for recursive constructors. We aim to satisfy the following rule:

$$\frac{\begin{array}{c}\Gamma_{\mathsf{spoke}} := (s : \mathrm{S}^1,\ f : \mathrm{S}^1 \to \|A\|_0,\ r_f : (s{:}\mathrm{S}^1) \to C[fs/t]) \\ M \in \|A\|_0\ [\Psi] \qquad a : A \gg P \in C[\mathsf{pt}(a)/t]\ [\Psi] \\ f : \mathrm{S}^1 \to \|A\|_0,\ r_f : (s{:}\mathrm{S}^1) \to C[fs/t] \gg H \in C[\mathsf{hub}(f)/t]\ [\Psi] \\ \Gamma_{\mathsf{spoke}} \gg O \in C[\mathsf{spoke}^z(s;f)/t]\ [\Psi,z] \qquad \Gamma_{\mathsf{spoke}} \gg O \doteq H \in C[\mathsf{spoke}^z(s;f)/t]\ [\Psi,z \mid z=0] \\ \Gamma_{\mathsf{spoke}} \gg O \doteq r_f s \in C[\mathsf{spoke}^z(s;f)/t]\ [\Psi,z \mid z=1]\end{array}}{\mathsf{trunc\text{-}elim}_{t.C}(M; a.P, f.H, z.s.f.O) \in C[M/a]\ [\Psi]}$$

The pt case P is completely standard. For constructors which take the recursive argument $f : \mathrm{S}^1 \to \|A\|_0$, we give access to the results $r_f : (s{:}\mathrm{S}^1) \to C[fs/t]$ of the recursive call: $r_f s$ stands for the result of calling $\mathsf{trunc\text{-}elim}$ on fs for each $s : \mathrm{S}^1$. As with the circle, we require that the endpoints of the path case line up with the corresponding boundary cases. Here, this means that the $z = 0$ endpoint of the $\mathsf{spoke}^z(s;f)$ case O must line up with the hub case H, while the $z = 1$ endpoint must be equal to the result $r_f s$ of the recursive call on fs. The operational semantics for $\mathsf{trunc\text{-}elim}$, which validates the above rule, again steps to the appropriate clause in each constructor case and uses a com in C to cover the fhcom case.

By definition, the type $\|A\|_0$ satisfies a canonicity theorem: any $M \in \|A\|_0\ [\Psi]$ evaluates to a value which is either a constructor (pt,hub, or spoke) or an fhcom. However, we also get a stronger guarantee if we only consider zero-dimension terms, that is, $M \in \|A\|_0\ [\varnothing]$. We have required that the set of constraints $\overrightarrow{\xi_i}$ in an fhcom tube is valid: there is some r such that either $(r = r) \in \overrightarrow{\xi_i}$ or both $(r = 0) \in \overrightarrow{\xi_i}$ and $(r = 1) \in \overrightarrow{\xi_i}$. In an empty dimension context, r must be either 0 or 1, in which case we know that one of the equations in $\overrightarrow{\xi_i}$ is true. Thus, any fhcom in an empty dimension context reduces. It follows that $M \in \|A\|_0\ [\varnothing]$ specifically evaluates to a constructor value. Moreover, neither hub nor spoke can be a value in an empty dimension context. We conclude that any $M \in \|A\|_0\ [\varnothing]$ evaluates to some $\mathsf{pt}(N)$: we can extract elements of A from zero-dimensional elements of $\|A\|_0$.

3.3 Identity Types

As our final example, we consider the identity type family specified in Figure 5, a type which has no path constructors but nonetheless requires new values in the higher-dimensional setting. While for the circle and truncation we needed new values to implement hcom, for the identity type we will need new values for coe. The new complication arises because the identity type is an *indexed inductive type*: a family $\mathsf{Id}_A(-,-)$ with a constructor refl which introduces elements at a particular index. These are distinct from *parameterized* inductive types, like $\|-\|_0$, whose constructors introduce elements uniformly at every index. In the indexed case, we need to account for coercion between indices of the family: if $P \in \mathsf{Path}_A(M,N)\ [\Psi]$, then to what should the term $\mathsf{coe}^{r \rightsquigarrow r'}_{z.\mathsf{Id}_A(M,P@z)}(\mathsf{refl}(M)) \in \mathsf{Id}_A(M,N)\ [\Psi]$ evaluate?

To define the denotation of indexed identity types, we need to simultaneously define a family of Ψ-relations as the least closed under certain operators. We thus introduce a notion of Δ-indexed Ψ-relations, where Δ is a context of Kan types.

Definition 3.2. Let $\Delta\ \mathsf{type}_{\mathsf{Kan}}\ [\Psi]$. A Δ-indexed Ψ-relation $\alpha = \alpha_-[-]$ consists of a Ψ'-relation $\alpha_\psi[\overline{I}]$ for every $\psi : \Psi' \to \Psi$ and $\overline{I} \in \Delta\psi'\ [\Psi']$, satisfying the following conditions.

(1) $\alpha_\psi[\overline{I}]\psi' = \alpha_{\psi\psi'}[\overline{I}\psi']$ for every $\psi' : \Psi'' \to \Psi'$,
(2) $\alpha_\psi[\overline{I}] = \alpha_\psi[\overline{I}']$ for every $\overline{I} \doteq \overline{I}' \in \Delta\psi\ [\Psi']$.

$$\frac{\overline{I} \neq \varnothing \qquad r \neq r'}{\mathsf{fcoe}^{r \rightsquigarrow r'}_{z.\overline{I}}(P)\ \mathsf{val}} \qquad \frac{\overline{I} \neq \varnothing}{\mathsf{fcoe}^{r \rightsquigarrow r}_{z.\overline{I}}(P) \longmapsto P} \qquad \frac{}{\mathsf{fcoe}^{r \rightsquigarrow r'}_{z.\varnothing}(P) \longmapsto P}$$

Fig. 9. Operational semantics of fcoe.

Note that a Δ-indexed Ψ-relation α is completely determined by the relations $\alpha_\psi[\overline{I}]_{\mathrm{id}}$.

We will define the identity family Id_A as the least (A, A)-indexed Ψ-relation closed under refl values, fhcom values, and finally fcoe values which will account for coercion between indices of a family. The operational semantics of fcoe are defined in Figure 9. The operator takes a line of indices $z.\overline{I}$, endpoints r, r', and an argument M to coerce from index $\overline{I}\langle r/z\rangle$ to $\overline{I}\langle r'/z\rangle$ of a family. As with fhcom, fcoe reduces when it is degenerate and is a value otherwise.

We define an operator on Δ-indexed Ψ-relations taking α to the PER of fcoe values built on α:

$$\textsc{Fcoe}(\alpha)_\psi[\overline{I}] := \left\{ \left(\mathsf{fcoe}^{r \rightsquigarrow r'}_{z.\overline{J}}(M), \mathsf{fcoe}^{r \rightsquigarrow r'}_{z.\overline{J}'}(M')\right) \;\middle|\; \begin{array}{l} \overline{J} \doteq \overline{J}' \in \Delta\psi\ [\Psi', z] \\ \overline{J}\langle r'/z\rangle \doteq \overline{I} \in \Delta\psi\ [\Psi'] \\ \textsc{Tm}(\textsc{Fcoe}(\alpha)_\psi[\overline{J}\langle r/z\rangle])(M, M') \end{array} \right\}$$

We then define $[\![\mathsf{Id}_A(M, N)]\!] := \beta[M, N]$, where β is the least fixed point of the operator $\textsc{Id}$ on (A, A)-indexed Ψ-relations given by

$$\begin{array}{rcl} \textsc{Id}(\alpha)_\psi[M, N] & := & \{(\mathsf{refl}(Q), \mathsf{refl}(Q')) \mid M \doteq N \doteq Q \doteq Q' \in A\psi\ [\Psi']\} \\ & & \cup\ \textsc{Fhcom}(\alpha_\psi[M, N]) \cup \textsc{Fcoe}(\alpha)_\psi[M, N]. \end{array}$$

Adding fcoe covers coercion between the indices of the family $\mathsf{Id}_A(-, -)$, but we still need to account for coercions $\mathsf{coe}^{r \rightsquigarrow r'}_{z.\mathsf{Id}_A(M,N)}$ where z occurs in the type argument A. To make up the difference, we introduce an auxiliary operator, the *total space coercion* $\mathsf{tcoe}^{r \rightsquigarrow r'}_{z.A}$, which we intend to satisfy the following typing rules:

$$\frac{P \in \mathsf{Id}_{A\langle r/z\rangle}(M, N)\ [\Psi]}{\begin{array}{c}\mathsf{tcoe}^{r \rightsquigarrow r'}_{z.A}(P) \in \mathsf{Id}_{A\langle r'/z\rangle}(\mathsf{coe}^{r \rightsquigarrow r'}_{z.A}(M), \mathsf{coe}^{r \rightsquigarrow r'}_{z.A}(N))\ [\Psi] \\ \mathsf{tcoe}^{r \rightsquigarrow r}_{z.A}(P) \doteq P \in \mathsf{Id}_{A\langle r/z\rangle}(M, N)\ [\Psi]\end{array}}$$

Total space coercion moves an element P along a path $z.A$ in the type argument of Id, carrying the endpoint indices $M, N \in A\langle r/z\rangle$ of the original path along via coercion. The name comes from homotopy theory: the *total space* is the pair type $(a, a'{:}A) \times \mathsf{Id}_A(a, a')$, and tcoe takes the triple

$$\langle M, N, P\rangle \in ((a, a'{:}A) \times \mathsf{Id}_A(a, a'))\langle r/z\rangle$$

to the triple

$$\langle \mathsf{coe}^{r \rightsquigarrow r'}_{z.A}(M), \mathsf{coe}^{r \rightsquigarrow r'}_{z.A}(N), \mathsf{tcoe}^{r \rightsquigarrow r'}_{z.A}(P)\rangle \in ((a, a'{:}A) \times \mathsf{Id}_A(a, a'))\langle r'/z\rangle.$$

Once we have implemented tcoe, we can combine it with fcoe to implement coe:

$$\mathsf{coe}^{r \rightsquigarrow r'}_{z.\mathsf{Id}_A(M,N)}(P) \longmapsto \mathsf{fcoe}^{r \rightsquigarrow r'}_{z.(\mathsf{coe}^{z \rightsquigarrow r'}_{z.A}(M), \mathsf{coe}^{z \rightsquigarrow r'}_{z.A}(N))}(\mathsf{tcoe}^{r \rightsquigarrow r'}_{z.A}(P)).$$

The inner tcoe produces an element of $\mathsf{Id}_{A\langle r'/z\rangle}(\mathsf{coe}^{r \rightsquigarrow r'}_{z.A}(M\langle r/z\rangle), \mathsf{coe}^{r \rightsquigarrow r'}_{z.A}(N\langle r/z\rangle))$; the outer fcoe massages the indices to turn this into an element of $\mathsf{Id}_{A\langle r'/z\rangle}(M\langle r'/z\rangle, N\langle r'/z\rangle)$.

To implement tcoe, we take the same approach as for coe in inductive types, evaluating the argument to a value and inspecting its form. The operational semantics of tcoe are displayed in Figure 10. On a refl value, tcoe becomes a coe in A applied to the argument; on fcoe and fhcom values, it pushes inside.

$$\frac{}{\mathsf{tcoe}^{r\rightsquigarrow r'}_{z.A}(\mathsf{refl}(M)) \longmapsto \mathsf{refl}(\mathsf{coe}^{r\rightsquigarrow r'}_{z.A}(M))}$$

$$\frac{}{\mathsf{tcoe}^{r\rightsquigarrow r'}_{z.A}(\mathsf{fcoe}^{s\rightsquigarrow s'}_{w.(M,N)}(P)) \longmapsto \mathsf{fcoe}^{s\rightsquigarrow s'}_{w.(\mathsf{coe}^{r\rightsquigarrow r'}_{z.A}(M),\mathsf{coe}^{r\rightsquigarrow r'}_{z.A}(N))}(\mathsf{tcoe}^{r\rightsquigarrow r'}_{z.A}(P))}$$

$$\frac{}{\mathsf{tcoe}^{r\rightsquigarrow r'}_{z.A}(\mathsf{fhcom}^{s\rightsquigarrow s}(M;\overrightarrow{\xi_i \hookrightarrow N_i})) \longmapsto \mathsf{fhcom}^{r\rightsquigarrow s'}(\mathsf{tcoe}^{r\rightsquigarrow r'}_{z.A}(M);\overrightarrow{\xi_i \hookrightarrow \mathsf{tcoe}^{r\rightsquigarrow r'}_{z.A}(N_i)})}$$

Fig. 10. Operational semantics of tcoe for the identity type.

For elimination, we obtain the standard J operator of intensional Martin-Löf type theory, i.e., the following typing rule.

$$\frac{\begin{array}{c} a,b : A, p : \mathsf{Id}_A(a,b) \gg C\ \mathsf{type}_{\mathsf{Kan}}\ [\Psi] \\ M,N \in A\ [\Psi] \qquad P \in \mathsf{Id}_A(M,N)\ [\Psi] \qquad a : A \gg R \in C[a/b][\mathsf{refl}(a)/p]\ [\Psi] \end{array}}{\mathsf{J}_{a.b.p.C}(M;N;P;a.R) \in C[M/a][N/b][P/p]\ [\Psi]}$$

When J is applied to the reflexive identity $\mathsf{refl}(Q)$, it steps to the provided case $R[Q/a]$. On fhcom terms, it takes the same tack as the circle and 0-truncation, stepping to a com in the target family. The fcoe case is similar, stepping to a coe in the target type as shown below.

$$\frac{F^z := \mathsf{fcoe}^{r\rightsquigarrow z}_{z.(M',N')}(P)}{\begin{array}{c}\mathsf{J}_{a.b.p.C}(M;N;\mathsf{fcoe}^{r\rightsquigarrow r'}_{z.(M',N')}(P);a.R) \longmapsto \\ \mathsf{coe}^{r\rightsquigarrow r'}_{z.C[M'/a][N'/b][F^z/p]}(\mathsf{J}_{a.b.p.C}(M'\langle r/z\rangle;N'\langle r/z\rangle;P;a.R))\end{array}}$$

Using J, we can define functions converting between the identity type $\mathsf{Id}_A(M,N)$ and the path type $\mathsf{Path}_A(M,N)$ as follows.

$$\lambda p.\ \mathsf{J}_{a.b.\_.\mathsf{Path}_A(a,b)}(M;N;p;a.\lambda^{\mathbb{I}}\_.a) \in \mathsf{Id}_A(M,N) \to \mathsf{Path}_A(M,N)$$
$$\lambda q.\ \mathsf{fcoe}^{0\rightsquigarrow 1}_{z.(M,q@z)}(\mathsf{refl}(M)) \in \mathsf{Path}_A(M,N) \to \mathsf{Id}_A(M,N)$$

It turns out that these functions form an equivalence, i.e., they are mutual inverses up to a path. We can thus use univalence to convert between theorems about Path and Id. However, the two types do not share the same *equality* properties. For example, with Id we have that $\mathsf{J}_{a.b.p.C}(M;N;\mathsf{refl}(Q);a.R) \doteq R[Q/a]$. If we transfer J across the equivalence to get a similar eliminator J' for Path, we find that it satisfies this equation only *up to a path*. Indeed, it appears to be impossible to write an eliminator for Path which satisfies J's typing rule and this equation. As such, while it is generally more convenient to work with Path, the type Id is necessary if we wish to interpret ITT in cubical type theory. (We discuss other approaches to this problem in Section 5.)

The definition of $\mathsf{Id}_A(M,N)$ implies that any $P \in \mathsf{Id}_A(M,N)\ [\Psi]$ evaluates either to a refl, fhcom, or fcoe value. As described in Section 3.2, we can rule out fhcom values when Ψ is empty. However, we cannot do the same for fcoe values, which should be no surprise: even in an empty context, $\mathsf{Path}_A(M,N)$ can be inhabited when M and N are not equal, and we know that $\mathsf{Id}_A(M,N)$ is inhabited whenever $\mathsf{Path}_A(M,N)$ is. Ultimately, we can only say that a zero-dimensional element of $\mathsf{Id}_A(M,N)$ is equal to a chain of fcoes applied to a refl term.

$\boxed{\text{Constructor lists: } \Delta \rhd \mathcal{K} \text{ constrs } [\Psi]}$

$$\frac{}{\Delta \rhd \bullet \text{ constrs } [\Psi]} \qquad \frac{\Delta \rhd \mathcal{K} \text{ constrs } [\Psi] \qquad \ell \notin \mathcal{K} \qquad \Delta \rhd \mathcal{K} \vdash C \text{ constr } [\Psi]}{\Delta \rhd [\mathcal{K}, \ell : C] \text{ constrs } [\Psi]}$$

Fig. 11. Definition of the constructor list judgment.

4 CUBICAL INDUCTIVE TYPES IN GENERALITY

In this section, we unify the three examples from Section 3 into a general schema for indexed cubical inductive types. We have already touched on each feature necessary for its implementation; the remaining task is to design the specification language. In this, we follow the standard pattern for inductive types: the arguments to a constructor can be chosen from a grammar of strictly positive type functions [Coquand and Paulin 1988; Dybjer 1994]. On top of that base, we add dimension parameters and a language of *boundary terms* for specifying the boundary of higher-dimensional constructors. The language of argument types and terms constitutes a small formal type theory which interprets into the computational type theory. A central complication of the higher case is that constructors are no longer independent of each other: each constructor can refer to previous constructors in its boundary.

4.1 The Schema

The central judgment $\Delta \rhd \mathcal{K}$ constrs $[\Psi]$ of our schema, defined in Figure 11, states that $\mathcal{K}$ is a labelled list of constructors for an inductive type indexed by Δ $\mathsf{ctx}_{\mathsf{Kan}}$ $[\Psi]$. This judgment is mutually inductively defined with a judgment $\Delta \rhd \mathcal{K} \vdash C$ constr $[\Psi]$ asserting that C is a constructor over a prefix $\mathcal{K}$. We draw labels ℓ from a fixed set L, writing $\ell \in \mathcal{K}$ to mean that ℓ occurs in $\mathcal{K}$ and $\mathcal{K}[\ell]$ for the constructor carrying label ℓ in $\mathcal{K}$.

The constructor judgment is itself mutually defined with judgments $\Delta \rhd \text{A} \equiv \text{A}'$ atype $[\Psi]$, $\Delta \rhd \Theta \equiv \Theta'$ actx $[\Psi]$, and $\Delta \rhd \mathcal{K}; \Theta \vdash \text{M} \equiv \text{M}' : \text{A}$ $[\Psi]$ which define the formal type theory of argument types A and boundary terms M.

Definition 4.1 (Constructors: $\Delta \rhd \mathcal{K} \vdash C$ constr $[\Psi]$)**.** Presupposing $\Delta \rhd \mathcal{K}$ constrs $[\Psi]$, we say $\Delta \rhd \mathcal{K} \vdash C$ constr $[\Psi]$ holds when $C = (\Gamma;\ \gamma.\overline{I};\ \gamma.\Theta;\ \overline{x}.\overrightarrow{\xi_k \hookrightarrow \gamma.\theta.\text{M}_k})$ where

(1) Γ $\mathsf{ctx}_{\mathsf{Kan}}$ $[\Psi]$,
(2) $\gamma : \Gamma \gg \overline{I} \in \Delta$ $[\Psi]$,
(3) $\gamma : \Gamma \gg \Delta \rhd \Theta$ actx $[\Psi]$,
(4) $\mathsf{FD}(\overrightarrow{\xi_k}) \subseteq \overline{x}$ and $\overrightarrow{\xi_k}$ is valid if non-empty,
(5) $\gamma : \Gamma \gg \Delta \rhd \mathcal{K}; \theta : \Theta \vdash \text{M}_k \equiv \text{M}'_l : \mathsf{X}(\overline{I})$ $[\Psi, \overline{x} \mid \xi_k, \xi_l]$ for each k, l.

In the informal notation used in Section 1, the data of Definition 4.1 corresponds to the following constructor entry.

$$\begin{array}{l} \text{data } \mathsf{X} \in \Delta \to \mathcal{U} \text{ where} \\ \quad \vdots \\ \quad \ell^{\overline{x}} \in (\gamma{:}\Gamma) \to \Theta \to \mathsf{X}(\overline{I})\ [\overrightarrow{\xi_i \hookrightarrow \text{M}_i}] \\ \quad \vdots \end{array}$$

The context Γ specifies the non-recursive parameters to the constructor, while the argument type Θ specifies the recursive arguments. The list of terms $\overline{I}$ specifies the index in Δ where the constructor lands, which can depend on Γ. The list $\overline{x}$ names the dimension parameters. Finally, the list of

$$\boxed{\text{Argument types: } \Delta \rhd \mathtt{A} \text{ atype } [\Psi]}$$

$$\frac{\overline{I} \in \Delta\ [\Psi]}{\Delta \rhd \mathsf{X}(\overline{I}) \text{ atype } [\Psi]} \qquad \frac{A \text{ type}_{\mathsf{Kan}}\ [\Psi] \qquad a : A \gg \Delta \rhd \mathtt{B} \text{ atype } [\Psi]}{\Delta \rhd (a{:}A) \to \mathtt{B} \text{ atype } [\Psi]}$$

$$\boxed{\text{Argument contexts: } \Delta \rhd \Theta \text{ actx } [\Psi]}$$

$$\frac{}{\Delta \rhd \varnothing \text{ actx } [\Psi]} \qquad \frac{\Delta \rhd \Theta \text{ actx } [\Psi] \qquad \Delta \rhd \mathtt{A} \text{ atype } [\Psi]}{\Delta \rhd (\Theta, p : \mathtt{A}) \text{ actx } [\Psi]}$$

Fig. 12. Definitions of the argument type and argument context judgments.

constraints and boundary terms $\overrightarrow{\xi_i \hookrightarrow \gamma.\theta.\mathtt{M}_i}$ specifies the boundary of the constructor, which can depend on both Γ and Θ. We leave it to the reader to infer the binary forms $\Delta \rhd \mathcal{K} \equiv \mathcal{K}'$ constrs $[\Psi]$ and $\Delta \rhd \mathcal{K} \vdash C \equiv C'$ constr $[\Psi]$ of the constructor judgments (or see [TR, 3.2]).

The argument type judgment $\Delta \rhd \mathtt{A}$ atype $[\Psi]$ is defined in Figure 12 (again, the reader may infer the binary form). The type $\mathsf{X}(\overline{I})$ is the recursive reference to index $\overline{I}$ of the inductive family being specified. On top of these types, we add dependent function types where the domain is an ordinary Kan type. Note that this judgment is inductively defined by rules, unlike the judgment $A \doteq A'$ type$_{\mathsf{pre}}$ $[\Psi]$ which is defined by evaluation. We use symbols $(\vdash, \equiv, :)$ rather than $(\gg, \doteq, \in)$ to emphasize this point. However, the open judgment $\Gamma \gg \Delta \rhd \mathtt{A} \equiv \mathtt{A}'$ atype $[\Psi]$ is still defined by functionality: we say that $\gamma : \Gamma \gg \Delta \rhd \mathtt{B} \equiv \mathtt{B}'$ atype $[\Psi]$ holds when $\Delta\psi[\overline{M}/\gamma] \rhd \mathtt{B}\psi[\overline{M}/\gamma] \equiv \mathtt{B}'\psi[\overline{M}'/a]$ atype $[\Psi']$ holds for every $\psi : \Psi' \to \Psi$ and $\overline{M} \doteq \overline{M}' \in \Gamma\psi\ [\Psi']$. The *argument context* judgment $\Delta \rhd \Theta$ actx $[\Psi]$, also inductively defined in Figure 12, is simply a list of argument types. Argument types can depend on ordinary terms, but not boundary terms, so there are no dependencies within such a context.

We define the well-typed *boundary terms* by a judgment $\Delta \rhd \mathcal{K}; \Theta \vdash \mathtt{M} \equiv \mathtt{M}' : \mathtt{A}\ [\Psi]$, which parameterized by a list $\mathcal{K}$ of previous constructors and an argument context Θ. This judgment is defined by the rules in Figure 13. Once again, this is an inductive definition; in particular, it is an inductive definition of an *open judgment* over a context Θ of argument variables. On the other hand, the "Γ-open" judgment $\gamma : \Gamma \gg \Delta \rhd \mathcal{K}; \Theta \vdash \mathtt{M} \equiv \mathtt{M}' : \mathtt{A}\ [\Psi]$ is still defined by functionality: it holds when

$$\Delta\psi[\overline{M}/\gamma] \rhd \mathcal{K}\psi[\overline{M}/\gamma]; \Theta\psi[\overline{M}/\gamma] \vdash \mathtt{M}\psi[\overline{M}/\gamma] \equiv \mathtt{M}'\psi[\overline{M}'/\gamma] : \mathtt{A}\psi[\overline{M}/\gamma]\ [\Psi']$$

holds for every $\psi : \Psi' \to \Psi$ and $\overline{M} \doteq \overline{M}' \in \Gamma\psi\ [\Psi']$.

We have access to three kinds of argument terms inhabiting the inductive family $\mathsf{X}(\overline{I})$: intro terms, representing constructors defined in $\mathcal{K}$, fcoe terms, and fhcom terms. Each of these is equipped with the expected boundary equations; in particular, the boundary of an intro term is that specified in its constructor data. The term $\mathsf{intro}_\ell^{\overline{r}}(\overline{P}; \overline{\mathtt{N}})$ takes a label ℓ pointing to its definition in $\mathcal{K}$, dimension parameters $\overline{r}$, non-recursive parameters $\overline{P}$, and recursive arguments $\overline{\mathtt{N}}$ according to its specification $\mathcal{K}[\ell]$. The function type $(a{:}A) \to \mathtt{B}$ is inhabited by λ-terms and supports elimination via application.

To realize instances of the schema, we first define interpretation functions taking argument types and boundary terms to "real" terms. For an argument type $\mathtt{B}$, we write $\{\!|\mathtt{B}|\!\}(\delta.A)$ for its interpretation where the indeterminant family X is instantiated by the type family $\delta.A$.

$$\begin{aligned} \{\!|\mathsf{X}(\overline{I})|\!\}(\delta.A) &:= A[\overline{I}/\delta] \\ \{\!|(b{:}B) \to \mathtt{C}|\!\}(\delta.A) &:= (b{:}B) \to \{\!|\mathtt{C}|\!\}(\delta.A). \end{aligned}$$

$\boxed{\text{Boundary terms: } \Delta \rhd \mathcal{K}; \Theta \vdash \mathrm{M} \equiv \mathrm{M}' : \mathrm{A}\ [\Psi]}$

Variables.

$$\frac{(p : \mathrm{A}) \in \Theta}{\Delta \rhd \mathcal{K}; \Theta \vdash p \equiv p : \mathrm{A}\ [\Psi]}$$

Constructors.

$$\frac{\mathcal{K}[\ell] = (\Gamma;\ \gamma.\overline{I};\ \gamma.\Phi;\ \overline{x}.\overrightarrow{\xi_k \hookrightarrow \gamma.\varphi.\mathrm{M}_k}) \quad \overline{r}\ \mathsf{dim}\ [\Psi] \qquad \overline{P} \in \Gamma\ [\Psi] \qquad \Delta \rhd \mathcal{K}; \Theta \vdash \overline{\mathrm{N}} : \Phi[\overline{P}/\gamma]\ [\Psi]}{\Delta \rhd \mathcal{K}; \Theta \vdash \mathsf{intro}_\ell^{\overline{r}}(\overline{P}; \overline{\mathrm{N}}) : \mathrm{X}(\overline{I}[\overline{P}/\gamma])\ [\Psi] \quad \Delta \rhd \mathcal{K}; \Theta \vdash \mathsf{intro}_\ell^{\overline{r}}(\overline{P}; \overline{\mathrm{N}}) \equiv \mathrm{M}_k\langle \overline{r}/\overline{x}\rangle[\overline{P}/\gamma][\overline{\mathrm{N}}/\varphi] : \mathrm{X}(\overline{I}[\overline{P}/\gamma])\ [\Psi \mid \xi_i]}$$

Coercion.

$$\frac{\overline{I} \in \Delta\ [\Psi, z] \qquad \Delta \rhd \mathcal{K}; \Theta \vdash \mathrm{M} : \mathrm{X}(\overline{I}\langle r/z\rangle)\ [\Psi]}{\Delta \rhd \mathcal{K}; \Theta \vdash \mathsf{fcoe}_{z.\overline{I}}^{r \rightsquigarrow r'}(\mathrm{M}) : \mathrm{X}(\overline{I}\langle r'/z\rangle)\ [\Psi] \quad \Delta \rhd \mathcal{K}; \Theta \vdash \mathsf{fcoe}_{z.\overline{I}}^{r \rightsquigarrow r'}(\mathrm{M}) \equiv \mathrm{M} : \mathrm{X}(\overline{I}\langle r'/z\rangle)\ [\Psi \mid r = r']}$$

$$\frac{\Delta \rhd \mathcal{K}; \Theta \vdash \mathrm{M} : \mathrm{X}(\varnothing)\ [\Psi]}{\Delta \rhd \mathcal{K}; \Theta \vdash \mathsf{fcoe}_{z.\varnothing}^{r \rightsquigarrow r'}(\mathrm{M}) \equiv \mathrm{M} : \mathrm{X}(\varnothing)\ [\Psi]}$$

Composition.

$$\frac{\Delta \rhd \mathcal{K}; \Theta \vdash \mathrm{M} : \mathrm{X}(\overline{I})\ [\Psi] \quad (\forall i, j)\ \Delta \rhd \mathcal{K}; \Theta \vdash \mathrm{N}_i \equiv \mathrm{N}_j : \mathrm{X}(\overline{I})\ [\Psi \mid \xi_i, \xi_j] \qquad (\forall i)\ \Delta \rhd \mathcal{K}; \Theta \vdash \mathrm{N}_i\langle r/y\rangle \equiv \mathrm{M} : \mathrm{X}(\overline{I})\ [\Psi \mid \xi_i]}{\begin{array}{c}\Delta \rhd \mathcal{K}; \Theta \vdash \mathsf{fhcom}_{\overline{I}}^{r \rightsquigarrow r'}(\mathrm{M}; \overrightarrow{\xi_i \hookrightarrow y.\mathrm{N}_i}) : \mathrm{X}(\overline{I})\ [\Psi] \\ \Delta \rhd \mathcal{K}; \Theta \vdash \mathsf{fhcom}_{\overline{J}}^{r \rightsquigarrow r'}(\mathrm{M}; \overrightarrow{\xi_i \hookrightarrow y.\mathrm{N}_i}) \equiv \mathrm{M} : \mathrm{X}(\overline{I})\ [\Psi \mid r = r'] \\ \Delta \rhd \mathcal{K}; \Theta \vdash \mathsf{fhcom}_{\overline{J}}^{r \rightsquigarrow r'}(\mathrm{M}; \overrightarrow{\xi_i \hookrightarrow y.\mathrm{N}_i}) \equiv \mathrm{N}_i\langle r'/y\rangle : \mathrm{X}(\overline{I})\ [\Psi \mid \xi_i]\end{array}}$$

Functions.

$$\frac{a : A \gg \Delta \rhd \mathcal{K}; \Theta \vdash \mathrm{N} : \mathrm{B}\ [\Psi]}{\Delta \rhd \mathcal{K}; \Theta \vdash \lambda a.\mathrm{N} : (a{:}A) \to \mathrm{B}\ [\Psi]} \qquad \frac{\Delta \rhd \mathcal{K}; \Theta \vdash \mathrm{N} : (a{:}A) \to \mathrm{B}\ [\Psi] \qquad M \in A\ [\Psi]}{\Delta \rhd \mathcal{K}; \Theta \vdash \mathsf{app}(\mathrm{N}; M) : \mathrm{B}[M/a]\ [\Psi]}$$

$$\frac{a : A \gg \Delta \rhd \mathcal{K}; \Theta \vdash \mathrm{N} : \mathrm{B}\ [\Psi] \qquad M \in A\ [\Psi]}{\Delta \rhd \mathcal{K}; \Theta \vdash \mathsf{app}(\lambda a.\mathrm{N}; M) \equiv \mathrm{N}[M/a] : \mathrm{B}\ [\Psi]}$$

$$\frac{\Delta \rhd \mathcal{K}; \Theta \vdash \mathrm{M} : (a{:}A) \to \mathrm{B}\ [\Psi]}{\Delta \rhd \mathcal{K}; \Theta \vdash \mathrm{M} \equiv \lambda a.(\mathsf{app}(\mathrm{M}; a)) : (a{:}A) \to \mathrm{B}\ [\Psi]}$$

Fig. 13. Definition of the boundary term typing judgment. We omit structural rules and congruence rules.

Similarly, we write $(\!|\theta.\mathrm{M}|\!)^{\mathcal{K}}(\overline{N})$ for the interpretation of an open argument term $\theta.\mathrm{M}$ in constructors $\mathcal{K}$ with terms $\overline{N}$ substituted for the variables Θ.

$$\begin{array}{rcl} (\!|\theta.\theta[j]|\!)^{\mathcal{K}}(\overline{N}) & := & \overline{N}[j] \\ (\!|\theta.\mathsf{intro}_\ell^{\overline{r}}(\overline{P}; \overline{\mathrm{N}})|\!)^{\mathcal{K}}(\overline{N}) & := & \mathsf{intro}_{\mathcal{K},\ell}^{\overline{r}}(\overline{P}; (\!|\theta.\overline{\mathrm{N}}|\!)^{\mathcal{K}}(\overline{N})) \\ & \vdots & \end{array}$$

$$\frac{\mathcal{K}[\ell] = (\Gamma;\ \gamma.\overline{I};\ \gamma.\Theta;\ \overline{x}.\overrightarrow{\xi_k \hookrightarrow \gamma.\theta.\textsc{m}_k}) \qquad (\forall k)\ \not\models \xi_k\langle \overline{r}/\overline{x}\rangle}{\mathsf{intro}^{\overline{r}}_{\mathcal{K},\ell}(\overline{P};\overline{N})\ \mathsf{val}}$$

$$\frac{\mathcal{K}[\ell] = (\Gamma;\ \gamma.\overline{I};\ \gamma.\Theta;\ \overline{x}.\overrightarrow{\xi_k \hookrightarrow \gamma.\theta.\textsc{m}_k}) \qquad \models \xi_k\langle \overline{r}/\overline{x}\rangle \qquad (\forall l < k)\ \not\models \xi_l\langle \overline{r}/\overline{x}\rangle}{\mathsf{intro}^{\overline{r}}_{\mathcal{K},\ell}(\overline{P};\overline{N}) \longmapsto (\!|\theta.\textsc{m}_k\langle \overline{r}/\overline{x}\rangle[\overline{P}/\gamma]|\!)^{\mathcal{K}}(\overline{N})}$$

Fig. 14. Operational semantics of intro.

$$\begin{aligned} mcoe^{r\rightsquigarrow r'}_{z.\varnothing}(\varnothing) &:= \varnothing \\ mcoe^{r\rightsquigarrow r'}_{z.(\gamma:\Gamma,a:A)}((\overline{M},M)) &:= (mcoe^{r\rightsquigarrow r'}_{z.\Gamma}(\overline{M}), \mathsf{coe}^{r\rightsquigarrow r'}_{z.A[mcoe^{r\rightsquigarrow z}_{z.\Gamma}(\overline{M})/\gamma]}(M)) \end{aligned}$$

Fig. 15. Definition of *mcoe*.

We leave it to the reader to infer the remaining clauses (or see [TR, 4.8]); we simply replace each boundary term with its corresponding ordinary term operator. In the intro clause, we add the constructor list $\mathcal{K}$ as an annotation, which is necessary for the operational semantics. In the $\mathsf{fhcom}_{\overline{I}}$ clause, we drop the index annotation, which is only included as a convenience for deriving the elimination rule.

With this definition in hand, we can give the operational semantics of intro terms, shown in Figure 14. When a boundary constraint holds, an intro steps to the interpretation of the corresponding boundary term applied to its arguments. Otherwise, it is a value.

Given a specification $\Delta \rhd \mathcal{K}$ constrs $[\Psi]$ for a cubical inductive type, we can construct the Δ-indexed Ψ-PER $[\![\mathsf{ind}_\Delta(\mathcal{K};-)]\!]$ as the least-fixed point of the operator on Δ-indexed Ψ-relations taking α to the indexed relation picking out the $\mathsf{intro}_{\mathcal{K},\ell}$ values for $\ell \in \mathcal{K}$, fcoe values, and fhcom values built on elements of $\textsc{Tm}(\alpha)$ [TR, 4.11]. The fact that this operator is monotone (and therefore has a least fixed-point) relies on the fact that all argument types $\textsc{a}$ represent strictly positive type operators.

Following the pattern of Section 3.1, we can then show that $\textsc{Tm}([\![\mathsf{ind}_\Delta(\mathcal{K};-)]\!])$ (applying $\textsc{Tm}$ pointwise to the indexed PER) is closed under $\mathsf{intro}_{\mathcal{K},\ell}$, fcoe, and fhcom terms [TR, 4.13-16].

4.2 Kan Operations

The definition of the Kan operations for the general case is essentially the sum of the techniques introduced in Section 3. Homogeneous composition is implemented by way of fhcom, while coercion is managed by fcoe and tcoe. The latter now takes the form $\mathsf{tcoe}^{r\rightsquigarrow r'}_{z.(\Delta,\mathcal{K})}$, transporting along lines in the indexing type and constructor data. To give the typing rule for tcoe, we introduce a *multi-coercion* meta-operation for coercing between contexts, defined in Figure 15: if $\overline{M} \in \Gamma\langle r/z\rangle\ [\Psi]$, then $mcoe^{r\rightsquigarrow r'}_{z.\Gamma}(\overline{M}) \in \Gamma\langle r'/z\rangle\ [\Psi]$. The general form of tcoe will satisfy the following typing rule [TR, §6.3].

$$\frac{P \in \mathsf{ind}_{\Delta\langle r/z\rangle}(\mathcal{K}\langle r/z\rangle;\overline{I})\ [\Psi]}{\mathsf{tcoe}^{r\rightsquigarrow r'}_{z.(\Delta,\mathcal{K})}(P) \in \mathsf{ind}_{\Delta\langle r'/z\rangle}(\mathcal{K}\langle r'/z\rangle; mcoe^{r\rightsquigarrow r'}_{z.\Delta}(\overline{I}))\ [\Psi]}$$

We follow the same pattern as in Section 3.3 to define the behavior of tcoe on fhcom and fcoe terms; the remaining task is to define tcoe on intro terms. For this, we divide into two cases: intro terms with and without boundaries. Recall that, in Definition 4.1, we required the list of boundary constraints for a constructor to be either empty or valid. We can effectively treat constructors in the former group as zero-dimensional constructors. For the latter group, we will need a "boundary-fixing" composite as in Section 3.2.

There is one additional complication in the general case, which arises from indexing but does not occur in the particularly simple case of the identity type. Much as we need a boundary-fixing fhcom when a constructor's boundary terms do not commute with coercion, we need an "index-fixing" fcoe when a constructor's index function ($\gamma.\overline{I}$ above) does not commute with coercion.

Suppose we want to apply $\mathsf{tcoe}^{r \rightsquigarrow r'}_{z.(\Delta,\mathcal{K})}$ to a term $\mathsf{intro}^{\overline{r}}_{\mathcal{K}\langle r/z\rangle,\ell}(\overline{P};\overline{N})$, where the associated constructor data is $\mathcal{K}[\ell] = (\Gamma;\ \gamma.\overline{I};\ \gamma.\Theta;\ \overline{x}.\overrightarrow{\xi_k \hookrightarrow \gamma.\theta.\mathrm{M}_k})$. Such a term lives in $\mathsf{ind}_{\Delta\langle r/z\rangle}(\mathcal{K}\langle r/z\rangle;\overline{I}\langle r/z\rangle[\overline{P}/\gamma])$, its index being determined by $\gamma.\overline{I}$ and the non-recursive parameters $\overline{P}$. If we simply coerce the arguments of the constructor and reapply $\mathsf{intro}_{\mathcal{K}\langle r'/z\rangle,\ell}$, we will obtain a term of type

$$\mathsf{ind}_{\Delta\langle r'/z\rangle}(\mathcal{K}\langle r'/z\rangle;\overline{I}\langle r'/z\rangle[\mathit{mcoe}^{r\rightsquigarrow r'}_{z.\Gamma}(\overline{P})/\gamma]).$$

However, our target typing rule for tcoe demands that we produce a term of type

$$\mathsf{ind}_{\Delta\langle r'/z\rangle}(\mathcal{K}\langle r'/z\rangle;\ \mathit{mcoe}^{r\rightsquigarrow r'}_{z.\Delta}(\overline{I}\langle r/z\rangle[\overline{P}/\gamma])).$$

Once again, we need to commute a coercion past the application of a function, and once again we can solve this by constructing an interpolating path:

$$\begin{array}{c}
\overline{I}\langle r'/z\rangle[\mathit{mcoe}^{r\rightsquigarrow r'}_{z.\Gamma}(\overline{P})/\gamma] \xrightarrow{\mathit{mcoe}^{y\rightsquigarrow r'}_{z.\Delta}(\overline{I}\langle y/z\rangle[\mathit{mcoe}^{r\rightsquigarrow y}_{z.\Delta}(\overline{P})/\gamma])} \mathit{mcoe}^{r\rightsquigarrow r'}_{z.\Delta}(\overline{I}\langle r/z\rangle[\overline{P}/\gamma]) \\
r' \xrightarrow{\qquad\qquad\qquad\qquad y \qquad\qquad\qquad\qquad} r
\end{array}$$

We can use an fcoe along this path to move the reconstructed intro term into the correct index. This is the only adjustment needed for the case of a constructor without boundary, which is covered by the rule marked (U) in Figure 16. For the case with boundary, we combine the boundary-fixing fhcom and index-fixing fcoe into a single fcom, which is defined from fcoe and fhcom as com is defined from coe and hcom:

$$\mathsf{fcom}^{r\rightsquigarrow r'}_{y.\overline{I}}(M;\overrightarrow{\xi_i \hookrightarrow y.N_i}) := \mathsf{fcom}^{r\rightsquigarrow r'}_{\overline{I}\langle r'/y\rangle}(\mathsf{fcoe}^{r\rightsquigarrow r'}_{y.\overline{I}}(M);\overrightarrow{\xi_i \hookrightarrow y.\mathsf{fcoe}^{y\rightsquigarrow r'}_{y.\overline{I}}(N_i)}).$$

This case is covered by the rule marked (B) in Figure 16. While these rules are notationally heavy, the idea is the same as in Section 3.2: coerce the arguments, then use the free Kan structure to adjust the index and boundary of the result.

4.3 Elimination

The general case brings no surprises for the eliminator; however, we will need to set up some machinery to state the operational semantics and typing rule. First, we define a grammar of *elimination lists* for specifying clauses.

$$\mathcal{E} ::= \bullet \mid [\mathcal{E},\ell : \overline{x}.\gamma.\eta.\rho.R] \text{ (where } |\eta| = |\rho|)$$

In each clause R, we have access to dimension parameters $\overline{x}$, non-recursive arguments γ, recursive arguments η, and results ρ (with $|\rho| = |\eta|$) of recursive calls on the variables in η. Once we define the eliminator list typing judgment $\Delta \rhd \mathcal{E} : \mathcal{K} \rightarrow \delta.h.D\ [\Psi]$, the typing rule for the eliminator will

$\boxed{\text{Free heterogeneous composition}}$

$$\frac{}{\mathsf{fcom}^{r\rightsquigarrow r'}_{z.\overline{I}}(M;\overrightarrow{\xi_i \hookrightarrow y.N_i}) \longmapsto \mathsf{fhcom}^{r\rightsquigarrow r'}(\mathsf{fcoe}^{r\rightsquigarrow r'}_{z.\overline{I}}(M);\overrightarrow{\xi_i \hookrightarrow y.\mathsf{fcoe}^{y\rightsquigarrow r'}_{z.\overline{I}}(N_i)})}$$

$\boxed{\text{Total space coercion}}$

$$\frac{\begin{array}{c}\mathcal{K}[\ell] = (\Gamma;\ \gamma.\overline{I};\ \gamma.\Theta;\ \overline{x}.\varnothing) \qquad \Theta = \overrightarrow{p_j : \mathrm{B}_j} \\ \overline{P}^z := \mathit{mcoe}^{r\rightsquigarrow z}_{z.\Gamma}(\overline{P}) \qquad (\forall j)\ N_j^z := \mathsf{coe}^{r\rightsquigarrow z}_{z.\{\!\!\{\mathrm{B}_j[\overline{P}^z/\gamma]\}\!\!\}(\delta.\mathsf{ind}_\Delta(\mathcal{K};\delta))}(N_j)\end{array}}{\mathsf{tcoe}^{r\rightsquigarrow r'}_{z.(\Delta,\mathcal{K})}(\mathsf{intro}^{\overline{r}}_{\mathcal{K}',\ell}(\overline{P};\overrightarrow{N_j})) \longmapsto \mathsf{fcoe}^{r'\rightsquigarrow r}_{z.\mathit{mcoe}^{z\rightsquigarrow r'}_{z.\Delta}(\overline{I}[\overline{P}^z/\gamma])}(\mathsf{intro}^{\overline{r}}_{\mathcal{K}\langle r'/z\rangle,\ell}(\overline{P}^{r'};\overrightarrow{N_j^{r'}}))}\ (\mathrm{U})$$

$$\frac{\begin{array}{c}\mathcal{K}[\ell] = (\Gamma;\ \gamma.\overline{I};\ \gamma.\Theta;\ \overline{x}.\overrightarrow{\xi_k \hookrightarrow \gamma.\theta.\mathrm{M}_k}) \qquad \overrightarrow{\xi_i} \neq \varnothing \qquad \Theta = \overrightarrow{p_j : \mathrm{B}_j} \qquad (\forall k)\ \not\models \xi_k\langle\overline{r}/\overline{x}\rangle \\ \overline{P}^z := \mathit{mcoe}^{r\rightsquigarrow z}_{z.\Gamma}(\overline{P}) \qquad (\forall j)\ N_j^z := \mathsf{coe}^{r\rightsquigarrow z}_{z.\{\!\!\{\mathrm{B}_j[\overline{P}^z/\gamma]\}\!\!\}(\gamma.\mathsf{ind}_\Delta(\mathcal{K};\gamma))}(N_j) \\ (\forall k)\ M_k^z := \mathsf{tcoe}^{z\rightsquigarrow r'}_{z.(\Delta,\mathcal{K})}((\!|\theta.\mathrm{M}_k\langle\overline{r}/\overline{x}\rangle[\overline{P}^z/\gamma]|\!)^{\mathcal{K}}(\overrightarrow{N_j^z}))\end{array}}{\begin{array}{c}\mathsf{tcoe}^{r\rightsquigarrow r'}_{z.(\Delta,\mathcal{K})}(\mathsf{intro}^{\overline{r}}_{\mathcal{K}',\ell}(\overline{P};\overrightarrow{N_j})) \qquad \longmapsto \\ \mathsf{fcom}^{r'\rightsquigarrow r}_{z.\mathit{mcoe}^{z\rightsquigarrow r'}_{z.\Delta}(\overline{I}[\overline{P}^z/\gamma])}(\mathsf{intro}^{\overline{r}}_{\mathcal{K}\langle r'/z\rangle,\ell}(\overline{P}^{r'};\overrightarrow{N_j^{r'}});\overrightarrow{\xi_k\langle\overline{r}/\overline{x}\rangle \hookrightarrow z.M_k^z})\end{array}}\ (\mathrm{B})$$

Fig. 16. Operational semantics of tcoe on intro terms.

have the following form [TR, §6.4.2].

$$\frac{\begin{array}{c}\Delta \rhd \mathcal{K}\ \mathsf{constrs}\ [\Psi] \qquad \delta : \Delta, h : \mathsf{ind}_\Delta(\mathcal{K};\delta) \gg D\ \mathsf{type}_{\mathsf{Kan}}\ [\Psi] \\ \overline{I} \in \Delta\ [\Psi] \qquad M \in \mathsf{ind}_\Delta(\mathcal{K};\overline{I})\ [\Psi] \qquad \Delta \rhd \mathcal{E} : \mathcal{K} \to \delta.h.D\ [\Psi]\end{array}}{\mathsf{elim}_{\delta.h.D;\overline{I}}(M;\mathcal{E}) \in D[\overline{I}/\delta][M/h]\ [\Psi]}$$

The eliminator expresses that the family $\mathsf{ind}_\Delta(\mathcal{K};-)$ is initial (in a suitable sense) among type families D which are algebras for the constructors named in $\mathcal{K}$. Given a clause for each constructor, here specified by $\mathcal{E}$, the eliminator takes any term M in index $\overline{I}$ of the inductive type to an element in index $\overline{I}$ of D.

To execute recursive calls on terms of compound argument type, we introduce an action of argument types on maps out of the inductive family.

$$\begin{array}{c}\mathit{act}_{\mathrm{X}(\overline{I})}(\delta.h.R;N) := R[\overline{I}/\delta][N/h] \\ \mathit{act}_{(b:B)\to\mathrm{C}}(\delta.h.R;N) := \lambda b.\mathit{act}_{\mathrm{C}}(\delta.h.R;\mathsf{app}(N,b))\end{array}$$

If $\delta : \Delta, h : \mathsf{ind}_\Delta(\mathcal{K};\delta) \gg R \in D\ [\Psi]$ is a map from the inductive family into a target family D and M is a term in $\{\!\!\{\mathrm{A}\}\!\!\}(\delta.\mathsf{ind}_\Delta(\mathcal{K};\delta))$, then $\mathit{act}_{\mathrm{A}}(\delta.h.R;N)$ is the result of applying $\delta.h.R$ in N at the leaves and recombining according to the shape of A. To express the type of $\mathit{act}_{\mathrm{A}}(\delta.h.R;N)$, we introduce the *dependent instantiation* of argument types.

$$\begin{array}{rcl}\{\!\!\{\mathrm{X}(\overline{I})\}\!\!\}_{\mathsf{d}}(\delta.h.D;N) & := & D[\overline{I}/\delta][N/h] \\ \{\!\!\{(b{:}B) \to \mathrm{C}\}\!\!\}_{\mathsf{d}}(\delta.h.D;N) & := & (b{:}B) \to \{\!\!\{\mathrm{C}\}\!\!\}_{\mathsf{d}}(\delta.h.D;\mathsf{app}(N,b)).\end{array}$$

We can then derive a typing rule for *act* [TR, 6.28].

$$\frac{\delta : \Delta, h : \mathsf{ind}_\Delta(\mathcal{K};\delta) \gg R \in D\ [\Psi]}{b : \{\!\!\{\mathrm{A}\}\!\!\}(\delta.\mathsf{ind}_\Delta(\mathcal{K};\delta)) \gg \mathit{act}_{\mathrm{A}}(\delta.h.R;b) \in \{\!\!\{\mathrm{A}\}\!\!\}_{\mathsf{d}}(\delta.h.D;b)\ [\Psi]}$$

With *act* in hand, we can state the operational semantics rule for elim on intro values.

$$\frac{\mathcal{K}[\ell] = (\Gamma;\ \gamma.\overline{J};\ \gamma.\Theta;\ \overline{x}.\overrightarrow{\xi_k \hookrightarrow \gamma.\theta.\mathrm{M}_k}) \qquad \mathcal{E}[\ell] = \overline{x}.\gamma.\eta.\rho.R \qquad \Theta = \overrightarrow{p_j : \mathrm{B}_j} \qquad (\forall k)\ \not\models \xi_k\langle\overline{r}/\overline{x}\rangle}{\mathsf{elim}_{\delta.h.D;\overline{I}}(\mathsf{intro}^{\overline{r}}_{\mathcal{K},\ell}(\overline{P};\overrightarrow{N_j});\mathcal{E}) \longmapsto R\langle\overline{r}/\overline{x}\rangle[\overline{P}/\gamma][\overrightarrow{N_j}/\eta][\overrightarrow{act_{\mathrm{B}_j[\overline{P}/\gamma]}(\delta.h.\mathsf{elim}_{\delta.h.D;\delta}(h;\mathcal{E});N_j)}/\rho]}$$

For each recursive argument N_j to the intro value with corresponding argument type B_j, we apply the action of B_j on the eliminator to N_j. The results are then supplied to the appropriate clause of $\mathcal{E}$. The operational semantics for elim on fcoe and fhcom terms matches that of the identity type case in Section 3.3: fcoes are taken to coes in the target family and fhcoms are taken to coms.

When we have a constructor with boundary, the elimination typing rule should require that the boundary of that constructor's case lines up with the cases for the constructor's boundary. To express this requirement, we need to compute the result of calling elim on a term $\theta.\mathrm{M}$ in terms of the results of calling elim on the variables in θ. Given a boundary term $\theta.\mathrm{M}$, terms $\overline{N}$, and terms $\overline{S}$ standing for the results of calling elim on $\overline{N}$, we define the *dependent term instantiation* $(\!|\theta.\mathrm{M}|\!)^{\mathcal{K},\mathcal{E}}_{\delta.h.D}(\overline{N};\overline{S})$ which combines the outputs $\overline{S}$ to produce the result of calling elim on $(\!|\theta.\mathrm{M}|\!)^{\mathcal{K}}(\overline{N})$.

$$\begin{array}{rcl}
(\!|\theta.\theta[j]|\!)^{\mathcal{K},\mathcal{E}}_{\delta.h.D}(\overline{N};\overline{S}) & := & \overline{S}[j] \\
(\!|\theta.\mathsf{intro}^{\overline{r}}_{\ell}(\overline{P};\overline{\mathrm{N}})|\!)^{\mathcal{K},\mathcal{E}}_{\delta.h.D}(\overline{N};\overline{S}) & := & R\langle\overline{r}/\overline{x}\rangle[\overline{P}/\gamma][(\!|\theta.\overline{\mathrm{N}}|\!)^{\mathcal{K}}(\overline{N})/\eta][(\!|\theta.\overline{\mathrm{N}}|\!)^{\mathcal{K},\mathcal{E}}_{\delta.h.D}(\overline{N};\overline{S})/\rho] \\
 & & \text{where } \mathcal{E}[\ell] = \overline{x}.\gamma.\eta.\rho.R \\
 & & \text{where } F^y = (\!|\theta.\mathsf{fhcom}^{r\rightsquigarrow y}_{\overline{I}}(\mathrm{M};\overrightarrow{\xi_i \hookrightarrow \mathrm{N}_i})|\!)^{\mathcal{K}}(\overline{N}) \\
(\!|\theta.\mathsf{fcoe}^{r\rightsquigarrow r'}_{z.\overline{I}}(\mathrm{M})|\!)^{\mathcal{K},\mathcal{E}}_{\delta.h.D}(\overline{N};\overline{S}) & := & \mathsf{coe}^{r\rightsquigarrow r'}_{z.D[\overline{I}/\delta][F^z/h]}((\!|\theta.\mathrm{M}|\!)^{\mathcal{K},\mathcal{E}}_{\delta.h.D}(\overline{N};\overline{S})) \\
 & & \text{where } F^z = (\!|\theta.\mathsf{fcoe}^{r\rightsquigarrow z}_{z.\overline{I}}(\mathrm{M})|\!)^{\mathcal{K}}(\overline{N}) \\
(\!|\theta.\lambda a.\mathrm{N}|\!)^{\mathcal{K},\mathcal{E}}_{\delta.h.D}(\overline{N};\overline{S}) & := & \lambda a.((\!|\theta.\mathrm{N}|\!)^{\mathcal{K},\mathcal{E}}_{\delta.h.D}(\overline{N};\overline{S})) \\
(\!|\theta.\mathsf{app}(\mathrm{N};M)|\!)^{\mathcal{K},\mathcal{E}}_{\delta.h.D}(\overline{N};\overline{S}) & := & \mathsf{app}((\!|\theta.\mathrm{N}|\!)^{\mathcal{K},\mathcal{E}}_{\delta.h.D}(\overline{N};\overline{S}),M) \\
 & \vdots &
\end{array}$$

(We have elided the notationally crowded fhcom clause; see [TR, 6.24].) The clauses for intro, fhcom, and fcoe match the operational semantics rules for elim, while the clause for λ matches the definition of *act*. We can show that this definition satisfies the following typing rule [TR, 6.27].

$$\frac{\Delta \rhd \mathcal{K};\Theta \vdash \mathrm{M} : \mathrm{A}\ [\Psi] \qquad \overline{N} \in \{\!|\Theta|\!\}(\delta.\mathsf{ind}_\Delta(\mathcal{K};\delta))\ [\Psi] \qquad \overline{S} \in \{\!|\Theta|\!\}_{\mathsf{d}}(\delta.h.D;\overline{N})\ [\Psi]}{(\!|\theta.\mathrm{M}|\!)^{\mathcal{K},\mathcal{E}}_{\delta.h.D}(\overline{N};\overline{S}) \in \{\!|\mathrm{A}|\!\}_{\mathsf{d}}(\delta.h.D;(\!|\theta.\mathrm{M}|\!)^{\mathcal{K}}(\overline{N}))\ [\Psi]}$$

Note that the type of $\overline{S}$ matches the type of $act_\Theta(\delta.h.\mathsf{elim}_{\delta.h.D;\delta}(h;\mathcal{E});\overline{N})$, while the type of the instantiation matches the type of $act_{\mathrm{A}}(\delta.h.\mathsf{elim}_{\delta.h.D;\delta}(h;\mathcal{E});(\!|\theta.\mathrm{M}|\!)^{\mathcal{K}}(\overline{N}))$.

We can now define the typing judgment $\Delta \rhd \mathcal{E} : \mathcal{K} \to \delta.h.D\ [\Psi]$ for elimination lists. Like constructor lists, these are built up inductively, so we first define a judgment $\Delta \rhd \mathcal{E} : \mathcal{K} \rightharpoonup \delta.h.D\ [\Psi]$ characterizing elimination lists defined on a prefix of a given $\mathcal{K}$. The definition is given in Figure 17. Here, dependent type instantiation is used to express the types of recursive calls ρ on the recursive arguments η, while the dependent term instantiation combines these recursive calls to produce the result $(\!|\theta.\mathrm{M}_k|\!)^{\mathcal{K},\mathcal{E}}_{\delta.h.D}(\eta;\rho)$ of calling elim on $(\!|\theta.\mathrm{M}|\!)^{\mathcal{K}}(\eta)$.

Finally, we define $\Delta \rhd \mathcal{K} : \mathcal{E} \to \delta.h.D\ [\Psi]$ to hold when $\Delta \rhd \mathcal{K} : \mathcal{E} \rightharpoonup \delta.h.D\ [\Psi]$ holds and $|\mathcal{E}| = |\mathcal{K}|$.

$\boxed{\text{Partial elimination lists: } \Delta \rhd \mathcal{E} : \mathcal{K} \rightharpoonup \delta.h.D\ [\Psi]}$

$$\frac{}{\Delta \rhd \bullet \equiv \bullet : \mathcal{K} \rightharpoonup \delta.h.D\ [\Psi]}$$

$$\frac{\begin{array}{c}\Delta \rhd \mathcal{E} : \mathcal{K} \rightharpoonup \delta.h.D\ [\Psi] \qquad \mathcal{K}[\ell] = (\Gamma;\ \gamma.\overline{I};\ \gamma.\Theta;\ \overline{x}.\overrightarrow{\xi_k \hookrightarrow \gamma.\theta.\mathrm{M}_k}) \\ \mathrm{ht}_{\mathcal{K}}(\ell) = |\mathcal{E}| \qquad \mathrm{H}_\ell := (\gamma : \Gamma, \eta : \{\!\!|\Theta|\!\!\}(\mathsf{ind}_\Delta(\mathcal{K};\overline{I})), \rho : \{\!\!|\Theta|\!\!\}_{\mathrm{d}}(\delta.h.D;\eta)) \\ \mathrm{H}_\ell \gg R \in D[\mathsf{intro}^{\overline{x}}_{\mathcal{K},\ell}(\gamma;\eta)/h]\ [\Psi,\overline{x}] \\ (\forall k)\ \mathrm{H}_\ell \gg R \doteq (\!|\theta.\mathrm{M}_k|\!)^{\mathcal{K},\mathcal{E}}_{\delta.h.D}(\eta;\rho) \in D[\mathsf{intro}^{\overline{x}}_{\mathcal{K},\ell}(\gamma;\eta)/h]\ [\Psi,\overline{x} \mid \xi_k]\end{array}}{\Delta \rhd [\mathcal{E}, \ell : \overline{x}.\gamma.\eta.\rho.R] : \mathcal{K} \rightharpoonup \delta.h.D\ [\Psi]}$$

Fig. 17. Typing rules for partial elimination lists. We write $\mathrm{ht}_{\mathcal{K}}(\ell)$ for the index at which ℓ appears in $\mathcal{K}$.

4.4 Canonicity

We obtain the usual canonicity guarantee in the general case: any $M \in \mathsf{ind}_\Delta(\mathcal{K};\overline{I})\ [\Psi]$ evaluates either to a constructor, an fcoe value, or an fhcom value. As with the examples, this is an immediate consequence of the definition of $\mathrm{T}\text{M}([\![\mathsf{ind}_\Delta(\mathcal{K};\overline{I})]\!])$. When Ψ is empty, the validity restriction on tubes allows us to exclude the case of an fhcom value or a constructor with boundary. For general indexed types, this is the best we can do. If, however, Δ is also empty, then we can exclude fcoe values, as we have defined fcoe to always reduce in this case. For non-indexed cubical inductive types, we thus have that any zero-dimensional element evaluates to a constructor without boundary.

5 RELATED WORK

Higher inductive types were first conceived by participants at the 2011 Oberwolfach workshop on homotopical interpretations of ITT. While an informal description of a general class was given in the HoTT Book [Univalent Foundations Program 2013, §6.13], the first rigorous presentation of a large syntactic class was Sojakova's W-*quotients* [Sojakova 2015], a generalization of W-types which added a path constructor. Sojakova showed that these types are *homotopy-initial algebras*, building on work on universal characterizations of ordinary inductive types in HoTT by Awodey et al. [2012]. More recently, Basold et al. [2017], Dybjer and Moeneclaey [2017], and Kaposi and Kovács [2018] have given schemata for higher inductive types which resemble ours, being described by a grammar or type theory of argument types and terms. The first accommodates 1-dimensional constructors, the second 2-constructors, and the third n-constructors as well as higher indexed and inductive-inductive types. Other work has focused on encoding complex HITs in terms of simpler ones [Kraus 2016; Rijke 2017; van Doorn 2016], but this is not possible for all HITs of interest [Lumsdaine and Shulman 2017, §9].

In comparison with this work, we benefit substantially from the cubical setting. First, we are able to handle n-constructors uniformly. 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, β-rules for path constructors only hold up to identity (see [Univalent Foundations Program 2013, §6.2]), which means that stating eliminators requires more and more path algebra as constructors refer to each other. Second, and more importantly, we are able to give an operational semantics with a strong canonicity theorem.

There are three main strands of cubical type theory: the substructural model (BCH) developed by Bezem et al. [2013], the cartesian type theory developed by Angiuli et al. [2017a,b, 2018] and the De Morgan type theory (CCHM) developed by Cohen et al. [2015]. These differ in the language of dimension terms: BCH treats dimension variables linearly, while CCHM adds *connection* and *reversal* operations on dimension terms. These variations also lead to different formulations of the Kan conditions. The linear dimension variables of BCH seem to create issues with proving elimination rules for higher inductive types. The other two theories, on the other hand, were each presented with examples of what we call CITs. Coquand et al. have expanded on their initial offering with further examples (such as pushouts and two approaches to the torus), sketched a schema, and proven consistency of these with a semantics in cubical sets [Coquand et al. 2018]. Our definitions of the values (including the use of fhcom), operational semantics, and rules for a non-indexed HIT specialize (roughly speaking) to the definitions given for these examples. The introduction of fcoe for the indexed case, however, is conceptually novel.

On the semantic side, Lumsdaine and Shulman [2017] define a class of HITs in certain simplicial model categories 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 fhcom and fcoe; our more fine-grained control seems to let us to sidestep the issue. For related reasons, they are not able to include compositions in boundary terms, whereas we can give access to fhcom and fcoe (but not tcoe). It is not clear to us whether our techniques can be adapted to their setting.

With regard to indexed inductive types, interest has mainly focused on the subproblem of adding identity types to cubical type theory. As described in Section 3.3, the native Path type in existing univalent cubical type theories appears not to support the J eliminator with its computation rule. The original work on resolving the issue is due to Swan [2014], who obtains an identity type in a category equivalent to the BCH model. Swan's original construction of Id is similar to ours, being defined as a restricted fibrant replacement, but it is not obvious how to adapt the construction to structural cubical type theories. Swan has now generalized the construction to a class of algebraic model structures using a cofibration-trivial fibration factorization [Swan 2018a]. Type-theoretically, this route may be viewed as defining

$$\mathsf{Id}_A(M,N) := (p{:}\mathsf{Path}_A(M,N)) \times \mathsf{Cyl}(A,\ (a,b{:}A) \times \mathsf{Path}_A(a,b),\ a.\langle a, a, \lambda^{\mathbb{I}}\_.a\rangle;\ \langle M,N,p\rangle),$$

where Cyl is the *mapping cylinder* indexed inductive type defined by Lumsdaine [2011, §2]. This is theoretically convenient: Cyl comes directly from the algebraic model structure, so one may make use of existing techniques for constructing such structures. However, it is unnecessarily indirect in this setting, where Cyl is no simpler to define than Id. Drawing inspiration from Swan, the CCHM theory defines identities as paths with labelled degeneracies that make it possible to distinguish reflexive paths; we believe that this construction could be transferred to a slight variation[1] of the type theory presented here. Finally, Swan has confirmed that Path fails to be an identity type in a range of constructive models of univalent type theory [Swan 2018b].

We conjecture that our indexed CITs can be implemented from the non-indexed fragment in the presence of an identity type, extending the encoding described by Altenkirch and Morris [2009, §6]. This encoding works by defining a non-indexed version of the indexed type where each constructor carries an additional argument expressing a self-reported index. An element of the indexed type then consists of (1) an element of the non-indexed type, and (2) a proof that its self-reported indices match those prescribed by the specification, the latter condition being formulated with Id. However, we expect this encoding to be computationally inefficient. A second approach would be to restrict our schema to constructors whose output indices (but not necessarily argument indices) are uniform. Such instances, which are also sometimes called parameterized inductive types, can be implemented without fcoe values. With an identity type, one can then

[1] The hcom-Kan condition must be modified to require that equality of hcom terms ignores false faces of the tube; for example, $\mathsf{hcom}_A^{0\rightsquigarrow 1}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i}, 0=1 \hookrightarrow y.P)$ must be equal to $\mathsf{hcom}_A^{0\rightsquigarrow 1}(M; \overrightarrow{\xi_i \hookrightarrow y.N_i})$.

encode indexed CITs by adding an equation on the output index (expressed with Id) as an extra argument to each constructor. We expect that this encoding—using either the Id we define, or the version from the CCHM theory—would be comparable in efficiency and general appearance to our direct approach; intuitively, the result is that index adjustments are stored in constructors rather than in separate fcoe wrappers. However, we believe that there is a conceptual advantage in treating Id as an instance of a general phenomenon, rather than as a special primitive. For either encoding, using Path rather than Id results in an eliminator without exact computation rules for constructors.

6 CONCLUSION

We have defined a schema for indexed cubical inductive types, defined the operational semantics of types so described, and shown that the computation system gives rise to a type theory which satisfies the expected introduction and elimination rules. We have analyzed the canonicity properties we can achieve for cubical inductive types, showing that in the non-indexed case we can ensure that all zero-dimensional values of an inductive type are constructors.

Our goal has been to design a schema which suffices to define the commonly-used examples of higher inductive types while remaing relatively simple. A number of incremental extensions appear to be possible. For example, it would be natural to allow dimension terms as indices to indexed inductive types. The language of argument types could also be extended. For instance, including Path types—which would require argument type dependency on boundary terms—would permit a direct definition of the 0-truncation without relying on the circle. Independently of the types, the language of boundary terms could also be extended. In particular, one could add the ability to define boundary terms by recursion on elements of previously defined CITs, which would be necessary to define the HIT described in [Lumsdaine and Shulman 2017, §9]. On a grander scale, one could follow Kaposi and Kovács and extend our schema to encompass the larger class of inductive-inductive (or inductive-recursive) types.

The non-indexed fragment of our schema has been implemented in two experimental proof assistants: **RedPRL**, a Nuprl-style proof refinement logic, and **redtt**, a high-level tactic language on top of a core type theory checked using normalization by evaluation [The RedPRL Development Team 2018a,b].

REFERENCES

Stuart F. Allen, Mark Bickford, Robert L. Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and E. Moran. 2006. Innovations in computational type theory using Nuprl. *J. Applied Logic* 4, 4 (2006), 428–469.

Thorsten Altenkirch and Peter Morris. 2009. Indexed Containers. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*. 277–285.

Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. 2017a. Cartesian Cubical Type Theory. (Dec. 2017). https://github.com/dlicata335/cart-cube.

Carlo Angiuli, Robert Harper, and Todd Wilson. 2017b. Computational higher-dimensional type theory. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*. 680–693.

Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. 2017c. Computational higher type theory III: Univalent universes and exact equality. (Dec. 2017). arXiv:1712.01800.

Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. 2018. Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities. In *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, United Kingdom.*

Steve Awodey, Nicola Gambino, and Kristina Sojakova. 2012. Inductive Types in Homotopy Type Theory. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*. 95–104.

Henning Basold, Herman Geuvers, and Niels van der Weide. 2017. Higher Inductive Types in Programming. *J. UCS* 23, 1 (2017), 63–88.

Marc Bezem, Thierry Coquand, and Simon Huber. 2013. A Model of Type Theory in Cubical Sets. In *19th International Conference on Types for Proofs and Programs, TYPES 2013, April 22-26, 2013, Toulouse, France*. 107–128.

Evan Cavallo and Robert Harper. 2018. Computational higher type theory IV: Inductive types. (Jan. 2018). arXiv:1801.01568.

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2015. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In *21st International Conference on Types for Proofs and Programs, TYPES 2015, May 18-21, 2015, Tallinn, Estonia*. 5:1–5:34.

Thierry Coquand, Simon Huber, and Anders Mörtberg. 2018. On Higher Inductive Types in Cubical Type Theory. In *33nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 9-12, 2018*.

Thierry Coquand and Christine Paulin. 1988. Inductively defined types. In *COLOG-88, International Conference on Computer Logic, Tallinn, USSR, December 1988, Proceedings*. 50–66.

Peter Dybjer. 1994. Inductive Families. *Formal Aspects of Computing* 6, 4 (1994), 440–465.

Peter Dybjer and Hugo Moeneclaey. 2017. Finitary Higher Inductive Types in the Groupoid Model. In *Mathematical Foundations of Programming Semantics, 33rd International Conference, Ljubljana, Slovenia*.

Martin Hofmann and Thomas Streicher. 1998. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*. Oxford Logic Guides, Vol. 36. Oxford Univ. Press, New York, 83–111.

Simon Huber. 2016. *Cubical Interpretations of Type Theory*. Ph.D. Dissertation. University of Gothenburg.

Ambrus Kaposi and András Kovács. 2018. A syntax for higher inductive-inductive types. In *3nd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*.

Nicolai Kraus. 2016. Constructions with Non-Recursive Higher Inductive Types. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*. 595–604.

Peter LeFanu Lumsdaine. 2011. Model Structures from Higher Inductive Types. (2011). Unpublished note. http://peterlefanulumsdaine.com/research/Lumsdaine-Model-strux-from-HITs.pdf.

Peter LeFanu Lumsdaine and Michael Shulman. 2017. Semantics of higher inductive types. (May 2017). arXiv:1705.07088.

Per Martin-Löf. 1975. An intuitionistic theory of types: predicative part. In *Logic Colloquium '73*, H.E. Rose and J.C. Shepherdson (Eds.). Studies in Logic and the Foundations of Mathematics, Vol. 80. North-Holland, 73–118.

Per Martin-Löf. 1982. Constructive Mathematics and Computer Programming. In *Logic, Methodology and Philosophy of Science*, L.J. Cohen, J. Łoś, H. Pfeiffer, and K.-P. Podewski (Eds.), Vol. VI. 153–175.

Egbert Rijke. 2017. The join construction. (Jan. 2017). arXiv:1701.07538.

Kristina Sojakova. 2015. Higher Inductive Types as Homotopy-Initial Algebras. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. 31–42.

Andrew Swan. 2014. An Algebraic Weak Factorisation System on 01-Substitution Sets: A Constructive Proof. (Sept. 2014). arXiv:1409.1829.

Andrew Swan. 2018a. Identity Types in Algebraic Model Structures and Cubical Sets. (Aug. 2018). arXiv:1808.00915.

Andrew Swan. 2018b. Separating Path and Identity Types in Presheaf Models of Univalent Type Theory. (Aug. 2018). arXiv:1808.00920.

The RedPRL Development Team. 2018a. RedPRL – the People's Refinement Logic. http://www.redprl.org/

The RedPRL Development Team. 2018b. `redtt`. https://github.com/RedPRL/redtt

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

Floris van Doorn. 2016. Constructing the propositional truncation using non-recursive HITs. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, Saint Petersburg, FL, USA, January 20-22, 2016*. 122–129.

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