Path spaces of higher inductive types in homotopy type theory

Kraus, Nicolai; von Raumer, Jakob

DOI:
10.1109/LICS.2019.8785661

License:
None: All rights reserved

Document Version
Peer reviewed version

Citation for published version (Harvard):
Kraus, N & von Raumer, J 2019, Path spaces of higher inductive types in homotopy type theory. in *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS).* IEEE Computer Society Press, Thirty-Fourth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2019), Vancouver, Canada, 24/06/19. https://doi.org/10.1109/LICS.2019.8785661

Path Spaces of Higher Inductive Types in Homotopy Type Theory

Nicolai Kraus
University of Nottingham
Eötvös Loránd University

Jakob von Raumer
University of Nottingham

Abstract—**The study of equality types is central to homotopy type theory. Characterizing these types is often tricky, and various strategies, such as the *encode-decode method*, have been developed.**

We prove a theorem about equality types of coequalizers and pushouts, reminiscent of an induction principle and without any restrictions on the truncation levels. This result makes it possible to reason directly about certain equality types and to streamline existing proofs by eliminating the necessity of auxiliary constructions.

To demonstrate this, we give a very short argument for the calculation of the fundamental group of the circle (Licata and Shulman [1]), and for the fact that pushouts preserve embeddings. Further, our development suggests a higher version of the Seifert-van Kampen theorem, and the set-truncation operator maps it to the standard Seifert-van Kampen theorem (due to Favonia and Shulman [2]).

We provide a formalization of the main technical results in the proof assistant Lean.

I. Introduction, Motivation, and Overview

A. Homotopy Type Theory

Martin-Löf's intensional type theory is a specific form of type theory which can serve as both a foundation for dependently typed programming languages and as a system in which mathematics can be developed. An important concept is *identity* or *equality types*: if A is a type and $x, y : A$ are elements, then $\mathsf{Id}_A(x, y)$ is a type whose elements we view as proofs that x and y are equal. Following a widespread convention, we denote this type by $(x =_A y)$ or $(x = y)$. In contrast, we denote definitions by $:\equiv$.

Homotopy type theory, commonly known as *HoTT*, is a variation of Martin-Löf's type theory. It is inspired by the observation that equalities behave like paths in homotopy theory, and this connection is so central that equality types are even referred to as *path spaces* in HoTT. As described in the book [3], two main features distinguish it from other variations of type theory. First, Voevodsky's *univalence axiom* (or *univalence principle*) ensures that the equality of types corresponds to equivalence of types ("coherent isomorphism"). Second, *higher inductive types* are an implementation of the idea that, if we can generate the elements of a type inductively, we could inductively generate its equalities at the same time.

Nicolai Kraus was supported by the Engineering and Physical Sciences Research Council (EPSRC), grant reference EP/M016994/1, and by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunication).

B. Quotients and Coequalizers

One central example for a class of higher inductive types is what we call *(homotopy) coequalizers* of relations. Coequalizers can (via straightforward constructions) be used to encode many other higher inductive types such as circles and spheres, tori, suspensions, general pushouts, and (via more difficult constructions) propositional truncations [4], [5] and higher truncations [6].

Coequalizers are of particularly great importance for the development of HoTT in the proof assistant *Lean* [7], since they are, together with propositional truncations, the only classes of higher inductive types that are defined in the prelude and thus "available by default". Much of HoTT in Lean is based on them, and they are known as *quotients* or *typal quotients* [8] in the Lean community. While they certainly *look* like quotients, we choose to avoid this name since it could be confusing for readers outside of the Lean community (see the discussion below).

Definition 1 (coequalizer of a relation)**.** Assume $A : \mathcal{U}$ is a type ($\mathcal{U}$ is a universe), and $\sim$ is a family of types indexed twice over A, sometimes called a "binary proof-relevant relation" $\sim : A \times A \to \mathcal{U}$; we write $(x \sim y)$ instead of $\sim (x, y)$. The *coequalizer* $A/\!\!/\!\sim$ is the higher inductive type generated by the constructors $[-]$ and glue, as in:

$$\begin{aligned} &\mathsf{data}\ A/\!\!/\!\sim\ : \mathcal{U} \\ &\quad [-] : A \to A/\!\!/\!\sim \\ &\quad \mathsf{glue} : \Pi\{a, b : A\}.(a \sim b) \to [a] = [b] \end{aligned} \tag{1}$$

The constructors express the idea that we take A and make related elements equal. We use curly brackets for the first two arguments of the glue constructor, $\{a, b : A\}$, to express that we will keep these arguments implicit to improve readability. On paper, we can view this as purely on the level of notation, i.e. write $\mathsf{glue}(s)$ simply as shorthand notation for $\mathsf{glue}(a, b, s)$.

Let us justify why we call $A/\!\!/\!\sim$ a *coequalizer*. In standard category theory, given two morphisms/functions $f, g : X \to A$, their coequalizer $\mathsf{Coequ}(f, g)$ can be thought of as the object/type A where $f(x)$ and $g(x)$ are identified. In "standard"

HoTT (as developed in the book [3]), this can be expressed as the following higher inductive type:

$$\begin{aligned} &\mathsf{data}\ \mathsf{Coequ}(f,g) : \mathcal{U} \\ &\quad \iota : A \to \mathsf{Coequ}(f,g) \\ &\quad \mathsf{resp} : (x : X) \to \iota(f(x)) = \iota(g(x)) \end{aligned} \tag{2}$$

Given f and g, we can define the relation $\sim$ on A by $(a \sim b) :\equiv \Sigma(x : X).(f(x) = a) \times (g(x) = b)$. It is then easy to see that $A/\!\!/\!\sim$ is equivalent to $\mathsf{Coequ}(f,g)$. In Lean, where the higher inductive type (2) is not available, we can thus use $A/\!\!/\!\sim$ instead.

Vice versa, if we start with a relation $\sim$, we can consider the two projections

$$\mathsf{proj}_1, \mathsf{proj}_2 : (\Sigma(a, b : A).a \sim b) \rightrightarrows A \tag{3}$$

Then, $\mathsf{Coequ}(\mathsf{proj}_1, \mathsf{proj}_1)$ is equivalent to $A/\!\!/\!\sim$.

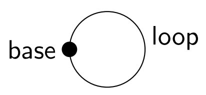
In order to explain why we choose to avoid calling $A/\!\!/\!\sim$ a quotient, we want to emphasize two points:

(I) Recall that a *(homotopy) set* in HoTT is a type satisfying the principle of unique identity proofs, i.e. a type A such that, for $a, b : A$ and $p, q : a = b$, we always have $p = q$. The type $A/\!\!/\!\sim$ is in general not a set. However, the variation of the construction which forces it to be a set is in the book [3] called a *set-quotient* and sometimes simply a *quotient*.

(II) The relation $\sim$ is neither required to be *(homotopy) propositional* (i.e. it is "proof relevant"), nor is it required to be reflexive, symmetric, or transitive.

It might be reasonable to speak of a quotient by a higher relation (cf. [9]) which is freely generated by $\sim$, but we do not go into this.

The points (I) and (II) above make coequalizers very flexible and remarkably powerful. Not forcing $A/\!\!/\!\sim$ to be a set lets us implement many interesting structures. For example, we can consider the (seemingly) trivial case where A is the unit type and $\sim$ is constantly unit as well. Then, the quantifications in the constructors in (1) are unnecessary, and (1) simplifies to the definition of the *circle type* $\mathbb{S}^1$, as on the right side:



$$\begin{aligned} &\mathsf{data}\ \mathbb{S}^1 : \mathcal{U} \\ &\quad \mathsf{base} : \mathbb{S}^1 \\ &\quad \mathsf{loop} : \mathsf{base} = \mathsf{base} \end{aligned} \tag{4}$$

The left side above shows how $\mathbb{S}^1$ can be drawn, thinking of elements as points and equalities as paths as suggested by the intuition that HoTT is inspired by.

Point (II) from above allows further important constructions. We have already seen that general (homotopy) coequalizers of two functions can be constructed. Similarly, the (homotopy) pushout P on the right can be defined as $(M + N)/\!\!/\!\sim$, where

$$\begin{tikzcd} L \arrow[r, "g"] \arrow[d, "f"'] & N \arrow[d, dashed] \\ M \arrow[r, dashed] & P \end{tikzcd}$$

$$(\mathsf{inl}(m) \sim \mathsf{inr}(n)) :\equiv \Sigma(l : L).(f(l) = m) \times (g(l) = n) \tag{5}$$

and $(\mathsf{inl}(m) \sim \mathsf{inl}(m'))$, $(\mathsf{inr}(n) \sim \mathsf{inr}(n'))$, $(\mathsf{inr}(n) \sim \mathsf{inl}(m))$ are all empty. Here, $\sim$ is neither reflexive, nor symmetric, nor transitive.

Higher inductive types, such as the ones above, allow the development of a synthetic version of homotopy theory inside HoTT (cf. [10], [11], [12], [13], [2], [14], [15], [16], [6]). A main objective of this line of research is to describe, classify, and compare path spaces (i.e. equality types) or homotopy groups (i.e. truncated path spaces) of higher inductive types such as circles and spheres.

For a (higher) inductive type, we know that its elements are generated by the constructors. This is expressed by *elimination principles*. Following the terminology of the book [3], we refer to the dependent elimination rule as *induction* and the non-dependent one as *recursion*. The induction principle for coequalizers, as it is standard in HoTT and implemented in Lean, states the following:

Principle 2 (induction for coequalizers)**.** Given a type family $P : A/\!\!/\!\sim \to \mathcal{U}$, and terms

$$f : \Pi(a : A).P([a]) \tag{6}$$

$$e : \Pi\{a, b : A\}, (s : a \sim b).f(a) =_{\mathsf{glue}(s)} f(b), \tag{7}$$

we get a term

$$\mathsf{ind}_{P,f,e} : \Pi(x : A/\!\!/\!\sim).P(x) \tag{8}$$

such that $\mathsf{ind}_{P,f,e}([a])$ computes to $f(a)$ and, if applied on $\mathsf{glue}(s)$, it equals what we get from e.

Here, we use the *path-over* a.k.a. *dependent path* notation [3, p.183] in the expression $f(a) =_{\mathsf{glue}(s)} f(b)$: Note that $f(a) : P([a])$ and $f(b) : P([b])$ do not have the same type, but by transporting/substituting along $\mathsf{glue}(s)$, we can equate them.

C. Motivation for the Main Result

Often, we want to find out what specific equality types look like. This is directly the goal when calculating the homotopy groups of given types (as in the synthetic homotopy theory mentioned above), but it is also a necessary intermediate step for many other constructions. For a very concrete example, let us recall the calculation of the loop space of the circle $\mathbb{S}^1$ by Licata and Shulman [1]. This loop space of $\mathbb{S}^1$, as defined above in (4), is by definition simply the equality type $(\mathsf{base} = \mathsf{base})$. Licata and Shulman introduce and explain the *encode-decode method*: in order go get started, they "guess" that the loop space in question is equivalent to the integers $\mathbb{Z}$ (looking at the left side of (4), the intuition is that one can go around the loop clockwise any number of times, and negative numbers correspond to going anticlockwise). Licata and Shulman then define a type family $\mathsf{Cover} : \mathbb{S}^1 \to \mathcal{U}$, inspired by the "guess", and construct functions between $\mathsf{Cover}(x)$ and $(\mathsf{base} = x)$ in order to show that these types are equivalent. Finally, observing that $\mathsf{Cover}(\mathsf{base})$ is $\mathbb{Z}$ gives the desired result.

The encode-decode method has been employed successfully in a variety of cases. Going through the necessary steps can be somewhat tedious but it often at least partially mechanical. One main goal in this paper is to develop a different method to directly work with equality types of coequalizers and pushouts (and constructions based on them): Since elimination rules such as Principle 2 characterize the points of an inductive type, and higher inductive types define points and equalities simultaneously, we believe that it is natural to hope for an "induction principle for equalities", i.e. a theorem which is reminiscent of an elimination rule. More concretely, for our case of coequalizers, let us assume we are given a type family

$$Q : \Pi\{a, b : A\}.([a] = [b]) \to \mathcal{U}. \tag{9}$$

We ask ourselves whether there are simple-to-check conditions that are sufficient to conclude $Q(q)$ for a general q, i.e. for any given $a, b : A$ with $q : [a] = [b]$.

Note that Q in (9) quantifies over two elements of A and an equality in $A/\!\!/\!\sim$. In comparison, if we asked the same question for a type family $S : \Pi(x, y : A/\!\!/\!\sim).(x = y) \to \mathcal{U}$ instead of Q, the answer would be the J eliminator (a.k.a. *path induction*), which says that it would be sufficient to prove $S(\mathsf{refl}_x)$. What we want and need in all the application is the version with "restricted" endpoints as in (9).

It turns out that there is an easy but powerful generalization of the above question. We get this generalization by switching from the *global* (or *unbased*) setting as in (9) to a *local* (*based*) one: we can fix one of the two endpoints to be $[a_0] : A/\!\!/\!\sim$ and replace Q by a family which is indexed only *once* over A,

$$P : \Pi\{b : A\}.([a_0] = [b]) \to \mathcal{U}. \tag{10}$$

This is akin to the difference between the standard J (a.k.a. *path induction*) and the Paulin-Mohring J [17] (a.k.a. *based path induction*). Just as for the two versions of J, a principle answering the based version of the question also answers the unbased one, and we thus focus exclusively on the former.

To get some intuition for the subtleties of equality types, let us first look at an "obvious" induction principle for (10) that turns out to be wrong. Usually, induction principles contain "one case for every constructor" (e.g. Principle 2 contains one case for $[-]$, and one case for glue). The standard equality constructor is refl, and (1) contains a further path constructor glue. Thus, we might try:

Incorrect principle. *Given a_0 and a family P as in* (10) *and terms*

$$r : P(\mathsf{refl}_{[a_0]}) \tag{11}$$
$$p : \Pi\{b : A\}, (s : a_0 \sim b).P(\mathsf{glue}(s)) \tag{12}$$

can we conclude $\Pi\{b : A\}, (q : [a_0] = [b]).P(q)$?

Counterexample. Consider the relation $\sim$ on the natural numbers $\mathbb{N}$, defined by $(m \sim n) :\equiv (m+1 = n)$. We can then look at the coequalizer $\mathbb{N}/\!\!/\!\sim$. Let us take $1 : \mathbb{N}$ as the base point and $P : \Pi(n : \mathbb{N}).([1] = [n]) \to \mathcal{U}$, defined by $P(n, q) :\equiv n \geq 1$. It is very easy to construct the terms r and p in (11,12). At the same time, we have that $P(0, \mathsf{glue}^{-1})$ is empty. □

The above naïve suggestion was easy to disprove, but let us try to understand why it was insufficient. Equalities that come from A can, by J, be assumed to be refl; these are sufficiently covered. However, this is not true for equalities that are generated using the glue constructor. The counterexample uses that we have not explicitly closed them under symmetry, and similarly, we could have used that we have not closed them under transitivity.

How could we fix this? Given an equality q in $A/\!\!/\!\sim$, we can compose it with $\mathsf{glue}(s)$ assuming the endpoints match, which suggests that the induction principle we are looking for should assume $Q(q) \to Q(q \cdot \mathsf{glue}(s))$, where $q \cdot p$ denotes the concatenation of two equalities p and q. We also can compose with $\mathsf{glue}(s)^{-1}$, suggesting that we also need $Q(q) \to Q(q \cdot \mathsf{glue}(s)^{-1})$. The operations of composing with $\mathsf{glue}(s)$ and composing with $\mathsf{glue}(s)^{-1}$ should be inverse to each other, which motivates us to ask for only *one* of them and require this one to be an equivalence, i.e. $Q(q) \simeq Q(q \cdot \mathsf{glue}(s))$. This leads us to a valid induction principle which is short, useful (as we will see when discussing applications), and comes with β-rules. Proving this principle is a main result of this paper:

Theorem 3 (induction for coequalizer equality)**.** *Assume we have A, $\sim$ as before and a point $a_0 : A$, and are further given a type family*

$$P : \Pi\{b : A\}.([a_0] = [b]) \to \mathcal{U} \tag{13}$$

together with terms

$$r : P(\mathsf{refl}_{[a_0]}) \tag{14}$$
$$e : \Pi\{b, c : A\}, (q : [a_0] = [b]), (s : b \sim c).$$
$$P(q) \simeq P(q \cdot \mathsf{glue}(s)) \tag{15}$$

Then, we can construct a term

$$\mathsf{ind}_{r,e} : \Pi\{b : A\}, (q : [a_0] = [b]).P(q) \tag{16}$$

with the following β-rules:

$$\mathsf{ind}_{r,e}(\mathsf{refl}_{[a_0]}) = r \tag{17}$$
$$\mathsf{ind}_{r,e}(q \cdot \mathsf{glue}(s)) = e(q, s, \mathsf{ind}_{r,e}(q)) \tag{18}$$

Remark 4. The above theorem can be proved in a way which makes the first β-rule (17) hold judgmentally. This is what we have done in our formalization (see Section I-G). It offers some additional convenience. In this paper, we do not track judgmental equalities explicitly.

D. Further Contents

Section II proves the stated theorems above. The proof makes use of the fact that such induction principles always have non-dependent counterparts which, when stated together with their uniqueness properties, are interderivable with the induction principles. The section should be seen as the core of the paper. It is split into three subsections introducing the main ideas, performing the technical constructions, and deriving the induction principle from the non-dependent version. We will see that it is useful to state our main results for pushouts instead of coequalizers, and this is discussed further in Section III. Building on this, we offer several further results and applications in this paper.

In Section IV, we demonstrate first applications of our theorems, one for the non-dependent coequalizer version and another for the dependent pushout version. Concretely, in Section IV-A, we show how our result *immediately* implies that the loop space of the circle (4) is equivalent to the integers [1]. In Section IV-B, we apply our result to prove a theorem in which our main result helps to avoid a somewhat tedious encode-decode agrument: Embeddings are preserved under pushouts [18].

The Seifert-van Kampen theorem, a result in algebraic topology allowing the computation of the fundamental group of a space if the groups of subspaces are known, was formulated and proved in HoTT by Favonia and Shulman [2]. How to do a *higher* version of this theorem, i.e. without set-truncation, is an open question in HoTT. In Section V, we suggest one formulation of a higher Seifert-van Kampen theorem and prove it using the main result of this paper.

E. Summary of our Contributions

- We prove a new induction principle for equality types in coequalizers and pushouts.
- To demonstrate the usefulness of this principle, we present a very short proof of the fact that the loop space of $\mathbb{S}^1$ is $\mathbb{Z}$ [1] and that embeddings are closed under pushout (possibly a new result in HoTT).
- We formulate and prove a higher dimensional Seifert-van Kampen theorem, generalizing the result by Favonia and Shulman [2].

F. Setting and Notation

We work in the "standard" version of homotopy type theory that is also developed in the book [3] with sums, Π- and Σ-types, a hierarchy of univalent universes $\mathcal{U}$ (we actually only need two), and inductive and higher inductive types. To be precise, the main part only requires *coequalizers* (Definition 1) and no other higher inductive types, which is what Lean provides by default; and our formalization does not require any further "higher inductive types via postulates". Only Section V makes use of the additional concept of an *indexed* higher inductive types, but this part is independent from the main results of this paper.

We use standard notation as used in [3] (with only minor modifications). In particular, we uncurry implicitly and write $f(a,b)$ instead of $f(a)(b)$ or $f_a(b)$ if f has type $A \to B \to C$. To further improve readability, we use *implicit arguments* (purely on the notational level) as explained after Definition 1 above.

G. Formalization

The main technical results have been formalized in the theorem prover Lean [7]: We formalized the equivalence of wild categories and constructed their initial objects as in Section II-A and II-B. We showed the non-dependent eliminator and its uniqueness, using a shortcut to make the first β-rule hold judgmentally, and used it to derive the dependent eliminator (the induction principle) with judgmental first β-rule as proved in Section II-C. We furthermore implemented the version for pushouts similar to the construction of Section III and proved that pushouts preserve embeddings (Section IV-B). We have not formalized the example in Section IV-A; the (in the context of this paper) interesting part of that example is immediate. Further, the development and discussion in Section V is not formalized.

Our code can be found at gitlab.com/fplab/freealgstr.

II. The Main Theorem: Path Spaces of Coequalizers

We will first formulate and prove the non-dependent version of the main result, by developing the corresponding categorical framework inside type theory. This then allows us to derive the induction principle as stated in Theorem 3.

A. Categorical Ideas in Type Theory

Using categorical ideas to structure constructions and reason inside type theory is standard. The induction (a.k.a. dependent elimination) principle of an inductive type can equivalently be formulated as a recursion (non-dependent elimination) principle together with a uniqueness principle, often formulated as a *universal property*. A principled way of doing this is to define objects and morphisms of a category; the statement is then that the inductive type in question is (homotopy) initial in this category. For the specific case of HoTT, the connection between induction and initiality has been shown by Awodey, Gambino and Sojakova [19] for inductive types, and by Sojakova [20] for some higher inductive types.

However, category theory in HoTT is subtle. The "obvious" naïve definition of a category without truncation (sometimes called a *wild category*; Definition 5) is not a well-behaved notion; for example, the slice of a wild category is not a wild category anymore. The underlying reason is that the identity and associativity equalities do not behave like laws, but like higher morphisms in a higher category where coherences are required. One approach to higher categories in HoTT is discussed in [21]. Alternatively, the *univalent categories* of [22] restrict the truncation levels to avoid the issue. For us, truncating is not a suitable strategy since it would not allow us to prove our general result.

Although not well-behaved in general, wild categories are still a useful tool in this paper. We do *not* think of them as "bad ordinary categories" but instead as an approximation to $(\infty,1)$-categories, where most of the (higher) data is omitted. However, since none of our constructions require us to actually *use* the omitted data, we get away with this. Most importantly, we can talk about the concept of (homotopy) initiality without ever referring to higher morphisms. Technically, we do not even need associativity; it could be excluded from the following definition without consequences for the rest of the paper.

Definition 5 (wild categories and initiality)**.** A *wild category* $\mathcal{A}$, for simplicity henceforth simply *category*, consists of a type $|\mathcal{A}|$ of objects; for $X, Y : |\mathcal{A}|$, a type $\mathcal{A}(X,Y)$ of morphisms; a composition operator $\circ$ and identities in the obvious way, together with the two standard equalities for the identies and one equality which states that $\circ$ is associative. An object X is called *initial* if, for every object Y, the type $\mathcal{A}(X,Y)$ is contractible (i.e. equivalent to the unit type).

For the whole section, let us assume that a type A together with $a_0 : A$ and a relation $\sim$ is given. Our main category of interest is the following:

Definition 6. The category $\mathcal{C}$ is defined as follows. Objects in $|\mathcal{C}|$ are "pointed type families respecting $\sim$", i.e. triples (K, r, e) of the types

$$K : A \to \mathcal{U} \tag{19}$$
$$r : K(a_0) \tag{20}$$
$$e : \Pi\{b, c : A\}.(b \sim c) \to K(b) \simeq K(c). \tag{21}$$

Morphisms are "pointed fibrewise functions". Explicitly, a morphism in $\mathcal{C}((K,r,e),(K',r',e'))$ is a triple (f,δ,γ):

$$f : \Pi(b : A).K(b) \to K'(b) \tag{22}$$
$$\delta : f_{a_0}(r) = r' \tag{23}$$
$$\gamma : \Pi\{b, c : A\},(s : b \sim c).e'(s) \circ f_b = f_c \circ e(s) \tag{24}$$

Here, γ is an equality witnessing that, for any $s : b \sim c$, the following square commutes:

$$\begin{tikzcd} K(b) \arrow[r, "e(s)"] \arrow[d, "f_b"'] & K(c) \arrow[d, "f_c"] \\ K'(b) \arrow[r, "e'(s)"] & K'(c) \end{tikzcd} \tag{25}$$

The remaining components (identities, composition, equations) are straightforward to define. For example, identities are given as $(\lambda b.\mathsf{id}, \mathsf{refl}_{r^i}, \lambda s.\mathsf{refl}_{e^i(s)})$ and composition by

$$(f', \delta', \gamma') \circ (f, \delta, \gamma) :\equiv (\lambda b.(f_b' \circ f_b), \mathsf{ap}_{f'_{a_0}}(\delta) \cdot \delta', \gamma' \circ \gamma), \tag{26}$$

where the last bit is given by pasting two vertically neighboring squares (25) (we do not think that writing down the full type-theoretic expression for this offers much insight).

A variation of Theorem 3, this time not as induction but as non-dependent elimination principle with uniqueness, can now be stated as follows:

Theorem 7 (initiality of coequalizer equality)**.** *Consider the object (K^i, p^i, e^i) of $\mathcal{C}$, where the first part is given by*

$$K^i(b) :\equiv ([a_0] = [b]), \tag{27}$$

i.e. K^i is given by equality in the coequalizer $A /\!\!/ \sim$. The point is given by

$$r^i :\equiv \mathsf{refl}_{[a_0]}. \tag{28}$$

For every $s : b \sim c$, the component $e^i(s)$ is the equivalence between $([a_0] = [b])$ and $([a_0] = [c])$ which is given by composition with $\mathsf{glue}(s)$; we simply write

$$e^i(s) :\equiv \_ \cdot \mathsf{glue}(s). \tag{29}$$

Then, our statement is: The object (K^i, p^i, e^i) is initial in the category $\mathcal{C}$.

Section II-B is devoted to the proof of this theorem, requiring various constructions and lemmas.

B. Initiality of Coequalizer Equality

In order to prove Theorem 7, we consider a second category which we call $\mathcal{D}$. We will then show that $\mathcal{C}$ and $\mathcal{D}$ are isomorphic. The point is that it is very easy to find the initial object in $\mathcal{D}$, and, via the isomorphism, it can easily be transported to $\mathcal{C}$. A useful technical tool is the formulation of coequalizer induction as an equivalence, which is what we start with.

Lemma 8 (coequalizer induction as equivalence)**.** *Given a type family $P : A /\!\!/ \sim \to \mathcal{U}$, there is a canonical map from the type*

$$\Pi(x : A /\!\!/ \sim).P(x) \tag{30}$$

to the type

$$\begin{aligned} &\Sigma(f : \Pi(a : A).P[a]). \\ &\quad \Pi\{a, b : A\},(s : a \sim b).f(a) =_{\mathsf{glue}(s)} f(b) \end{aligned} \tag{31}$$

mapping g to the pair $(g \circ [-], \lambda s.\mathsf{apd}_g(\mathsf{glue}(s)))$. This canonical map is an equivalence.

Note that apd is the "dependent ap function" [3].

Proof. The standard induction principle, given as Principle 2 above, states that there is a function from (31) to (30) with β-rules that essentially amount to stating that this function is a section of the canonical map above. Lemma 8 replaces "section" by "inverse". This easily follows from the standard induction principle. We are not the first to make this observation: a small variation of the lemma is already present in the Lean library [23]. ☐

Remark 9. Note that Lemma 8 crucially depends on the "non-recursiveness" of $A /\!\!/ \sim$. For example, the analogous statement for the natural numbers $\mathbb{N}$ is false (i.e. assuming it leads to a contradiction).

In line with the strategy outlined above, we further consider the following category $\mathcal{D}$:

Definition 10 (category $\mathcal{D}$)**.** $\mathcal{D}$ is the category of pointed families over $A /\!\!/ \sim$. Explicitly, objects in $|\mathcal{D}|$ are pairs (L, p) as in

$$L : A /\!\!/ \sim \to \mathcal{U} \tag{32}$$
$$p : L([a_0]), \tag{33}$$

and morphisms in $\mathcal{D}((L,p),(L',p'))$ are pairs (g,ϵ) of types

$$g : \Pi(x : A/\!\!/\!\sim).L(x) \to L'(x) \tag{34}$$

$$\epsilon : g(p) = p' \tag{35}$$

Again, the remaining components of the category are defined in the straightforward way.

The connection between $\mathcal{C}$ and $\mathcal{D}$ is as follows:

Lemma 11. *The two categories are isomorphic, in the following sense. There is a map*

$$\Phi_0 : |\mathcal{D}| \to |\mathcal{C}| \tag{36}$$

which is an equivalence, and there is also a map

$$\Phi_1 : \Pi(X, Y : |\mathcal{D}|).\mathcal{D}(X,Y) \to \mathcal{C}(\Phi_0(X), \Phi_0(Y)) \tag{37}$$

such that each $\Phi_1(X,Y)$ is an equivalence. Moreover, identities and compositions are preserved by the equivalence.

Proof. Let us unfold the type in (36); this is the type of the equivalence Φ_0 that we *want* to construct:

$$\begin{aligned} &\Sigma(L : A/\!\!/\!\sim \to \mathcal{U}).L([a_0]) \\ \simeq\quad &\Sigma(K : A \to \mathcal{U}).\Sigma(p : K(a_0)). \\ &\quad e : \Pi\{b,c : A\},(s : b \sim c).K(b) \simeq K(c) \end{aligned} \tag{38}$$

Lemma 8 gives us a tool to construct equivalences. Let us use that lemma with the constant family $P(x) :\equiv \mathcal{U}$; this makes use of the fact that the lemma works on all universe levels. The lemma then gives us, simply by replacing $P(x)$ by $\mathcal{U}$, renaming variables, and using that we are now in the non-dependent special case, the following equivalence φ_0:

$$\begin{aligned} &(A/\!\!/\!\sim \to \mathcal{U}) \\ \simeq\quad &\Sigma(K : A \to \mathcal{U}). \\ &\quad e : \Pi\{b,c : A\},(s : b \sim c).K(b) = K(c) \end{aligned} \tag{39}$$

Moreover, we know how φ_0 is defined, namely by

$$\varphi_0(L) :\equiv (L \circ [-], \lambda s.\mathsf{ap}_L(\mathsf{glue}(s))) \tag{40}$$

(since we are in the non-dependent case, apd became ap).

We claim that the function Φ_0 of type (38) can be constructed from the function φ_0 of type (39) via two small modifications:

- First, if we compare the domains of Φ_0 with the domain of φ_0, and the codomain of Φ_0 with the codomain of φ_0, we see that the "point-component" is missing from φ_0, i.e. the Σ-component $L([a_0])$ is missing in its domain and $(p : K(a_0))$ is missing in its codomain. However, we can just extend domain and codomain with this Σ-component. The equation (40) tells us that this extension is completely trivial, since $K \equiv L \circ [-]$, i.e. we extend φ_0 with the identity on one additional new component.
- The codomain of this extended φ_0 only differs from the codomain of Φ_0 in that the component e in (39) concludes with $(K(b) = K(c))$, while the component e in (38) concludes with $(K(b) \simeq K(c))$. To close this gap, we can use the canonical function idtoeqv which turns an equality between types into an equivalence (cf. [3]), and of which the univalence axiom ensures that it is an equivalence itself.

This concludes the construction of the equivalence Φ_0, and, using (40), we can write down how the function part of it computes when applied to a pair (L,p):

$$\Phi_0(L,p) \equiv \big(L \circ [-], p, \lambda s.\mathsf{idtoeqv}(\mathsf{ap}_L(\mathsf{glue}(s)))\big) \tag{41}$$

The construction of Φ_1 as in (37) is slightly more subtle since it depends on Φ_0, but works in essentially the same way. Assume we are given (L,p) and (L',p') in $|\mathcal{D}|$. We unfold the type of $\Phi_1((L,p),(L',p'))$ as in (37), making use of equation (41). This gives us the type that we *want* to inhabit:

$$\begin{aligned} &\Sigma\,(g : \Pi(x : A/\!\!/\!\sim).L(x) \to L'(x))\,. \\ &\quad \epsilon : g(p) = p' \\ \simeq\quad &\Sigma\,(f : \Pi(b : A).L([b]) \to L'([b]))\,. \\ &\Sigma(\delta : f(p) = p'). \\ &\quad \gamma : \Pi\{b,c : A\},(s : b \sim c). \\ &\qquad\qquad \mathsf{idtoeqv}(\mathsf{ap}_L(\mathsf{glue}(s))) \circ f(b) \\ &\qquad\qquad = f(c) \circ \mathsf{idtoeqv}(\mathsf{ap}_{L'}(\mathsf{glue}(s))) \end{aligned} \tag{42}$$

Let us use Lemma 8 again, this time with the family $P(x) :\equiv (L(x) \to L'(x))$. Simply by plugging this into Lemma 8 (and renaming variables), we get the following equivalence φ_1:

$$\begin{aligned} &(\Pi(x : A/\!\!/\!\sim).L(x) \to L'(x)) \\ \simeq\quad &\Sigma(f : \Pi(b : A).L([b]) \to L'([b])). \\ &\quad \gamma : \Pi\{b,c : A\},(s : b \sim c).f(b) =_{\mathsf{glue}(s)} f(c) \end{aligned} \tag{43}$$

Similar to what we have done before, we have to use (43) to derive (42); and as before, there are two steps. First, we need to add the equation for the points (i.e. the components ϵ and δ), but this is as simple and direct as before; we do not spell out the details.

Second, and more interestingly, we have to show that the γ's of (42) and (43) coincide (i.e. that their types are equivalent). As very often in HoTT when we want to prove something for a specific equality (here $\mathsf{glue}(s)$), the easiest way to do this is to generalize the statement and formulate it in terms of an *arbitrary* equality, which then allows path induction. The only red herring here is that f is a family of functions; but, since it is indexed over A and the equality in question lives in $A/\!\!/\!\sim$, we cannot make use of this. The equivalence follows from Lemma 12 below, by using $f(b)$ for h, and $f(c)$ for k, and $\mathsf{glue}(s)$ for q.

It is easy to check that Φ_1 preserves identities and compositions of morphisms. ☐

Lemma 12. *Let Z be a type, $F, G : Z \to \mathcal{U}$ two type families, $x, y : Z$ and $q : x = y$ elements and an equality. Assume we have functions $h : F(x) \to G(x)$ and $k : F(y) \to G(y)$. Then, the type $(h =_q k)$ is equivalent to the type*

$$\mathsf{idtoeqv}(\mathsf{ap}_G(q)) \circ h = k \circ \mathsf{idtoeqv}(\mathsf{ap}_F(q)). \tag{44}$$

Proof. By induction, we can assume $q \equiv \mathsf{refl}$, in which case both expressions evaluate to $(h = k)$. ☐

Having shown Lemma 11, which constitutes the main technical difficulty of the proof of Theorem 7, we can work with $\mathcal{D}$ instead of $\mathcal{C}$. The benefit is that it is easy to find the initial object of $\mathcal{D}$:

Lemma 13. *Let us consider the object (L^i, p^i) of $\mathcal{D}$, given as follows:*

$$L^i(x) :\equiv ([a_0] = x) \tag{45}$$
$$p^i :\equiv \mathsf{refl}_{[a_0]}. \tag{46}$$

This object is initial in $\mathcal{D}$.

Proof. Let (L, p) be any other object. After unfolding the definition in (34,35), the type $\mathcal{D}((L^i, p^i), (L, p))$ is given by

$$\begin{aligned}&\Sigma\big(g : \Pi(x : A/\!\!/\!\sim).([a_0] = x) \to L(x)\big).\\ &\quad \epsilon : g([a_0], \mathsf{refl}) = p\end{aligned} \tag{47}$$

This type is contractible by applying "singleton contraction" twice: first, we use that an element x together with an equality $[a_0] = x$ form a contractible pair, simplifying the above type to $\Sigma(g : L([a_0]).g = p$; and this type is clearly contractible. □

Having found the initial object in $\mathcal{D}$, we transport it to $\mathcal{C}$ in order to prove the categorical version of our main result, namely Theorem 7:

Proof of Theorem 7. Since Φ_1 as constructed in Lemma 11 preserves morphism spaces, Φ_0 preserves the initial object. Thus, all we need to do is to use the object found in Lemma 13 and compute using (41). This gives us K_0^i and r_0^i immediately. The last component e_0^i is correct by a standard "path induction"-argument. □

C. Derivation of the Induction Principle

The main part of the derivation of the based induction principle (Theorem 3) from the non-dependent based formulation (Theorem 7) is completely standard and follows known principles, cf. the work by Awodey, Gambino, and Sojakova [19]. We use the "total space" construction to turn the dependent case into the non-dependent one. Afterwards, we still need to derive the β-rules, and this is trickier; we use a small trick to "strictify" equations. Let us restate the theorem which we want to prove:

Theorem 3 (induction for coequalizer equality)**.** *Assume we have A, $\sim$ as before and a point $a_0 : A$, and are further given a type family*

$$P : \Pi\{b : A\}.([a_0] = [b]) \to \mathcal{U} \tag{13}$$

together with terms

$$r : P(\mathsf{refl}_{[a_0]}) \tag{14}$$
$$\begin{aligned}e : \Pi\{b, c : A\}, (q : [a_0] = [b]), (s : b \sim c).\\ P(q) \simeq P(q \cdot \mathsf{glue}(s))\end{aligned} \tag{15}$$

Then, we can construct a term

$$\mathsf{ind}_{r,e} : \Pi\{b : A\}, (q : [a_0] = [b]).P(q) \tag{16}$$

with the following β-rules:

$$\mathsf{ind}_{r,e}(\mathsf{refl}_{[a_0]}) = r \tag{17}$$
$$\mathsf{ind}_{r,e}(q \cdot \mathsf{glue}(s)) = e(q, s, \mathsf{ind}_{r,e}(q)) \tag{18}$$

Proof. Assume P, r and e are given. The "total space" versions of these three components form an object $(\overline{P}, \overline{r}, \overline{e})$ of the category $\mathcal{C}$, and they are defined as follows:

$$\overline{P} : A \to \mathcal{U} \tag{48}$$
$$\overline{P}(b) :\equiv \Sigma(q : [a_0] = [b]).P(q) \tag{49}$$
$$\overline{r} : \overline{P}(a_0) \tag{50}$$
$$\overline{r} :\equiv (\mathsf{refl}_{[a_0]}, r) \tag{51}$$
$$\overline{e} : \Pi\{b, c : A\}.(b \sim c) \to \overline{P}(b) \simeq \overline{P}(c) \tag{52}$$
$$\overline{e}(s) :\equiv (\_ \cdot \mathsf{glue}(s), e(\_, s, \_))\,. \tag{53}$$

Note that the last line (53) implicitly uses that an equivalence between Σ-types can be constructed from a pair of equivalences for the first and second component. Explicitly, the function part of the equivalence $\overline{e}(s)$ maps a given pair (q, x) with $q : [a_0] = [b]$ and $x : P(q)$ to the pair $(q \cdot \mathsf{glue}(s), e(q, s, x))$.

We have a morphism from the initial object of $\mathcal{C}$ to this newly constructed object (let's call it (f, δ, γ)), but we also have the "first projection" into the other direction:

$$(f, \delta, \gamma) : \mathcal{C}\big((K^i, p^i, e^i), (\overline{P}, \overline{r}, \overline{e})\big) \tag{54}$$
$$(\lambda b.\mathsf{proj}_1, \mathsf{refl}_{r^i}, \lambda s.\mathsf{refl}_{e^i(s)}) : \mathcal{C}\big((\overline{P}, \overline{r}, \overline{e}), (K^i, p^i, e^i)\big) \tag{55}$$

It follows from initiality that the composition of these morphisms is the identity on the object (K^i, p^i, e^i), i.e. we have a ψ of the following type:

$$\begin{aligned}\psi :\ &(\lambda b.\mathsf{proj}_1, \mathsf{refl}_{r^i}, \lambda s.\mathsf{refl}_{e^i(s)}) \circ (f, \delta, \gamma)\\ &\qquad = (\lambda b.\mathsf{id}, \mathsf{refl}_{r^i}, \lambda s.\mathsf{refl}_{e^i(s)})\end{aligned} \tag{56}$$

In particular, given any $q : [a_0] = [b]$, we get an equality

$$\psi_q^1 : \mathsf{proj}_1(f_b(q)) = q \tag{57}$$

and we can define:

$$\mathsf{ind}_{r,e}(q) : P(q) \tag{58}$$
$$\mathsf{ind}_{r,e}(q) :\equiv \mathsf{transport}^P(\psi_q^1, \mathsf{proj}_2(f_b(q))). \tag{59}$$

This defines the induction principle, but the two β-rules still need to be justified. The equality ψ in (56) consists of three parts, one for each component [3, Thm 2.7.2]; let us write (ψ^1, ψ^2, ψ^3) for them. The general idea is that, just as ψ^1 has allowed us to construct the induction principle (59), ψ^2 allows us to show the first β-equation and ψ^3 gives us the second. The main difficulty here are the many *transports/pathovers* involved, since the types of ψ^2 and ψ^3 depend on ψ^1. The trick is to split f into (f^1, f^2) by setting $f_b^1 :\equiv \mathsf{proj}_1 \circ f_b$, $f_b^2 :\equiv \mathsf{proj}_2 \circ f_b$, and similarly split δ and γ. Using this, and calculating the left side of (56), we get

$$(\psi^1, \psi^2, \psi^3) :\ (f^1, \delta^1, \gamma^1) = (\lambda b.\mathsf{id}, \mathsf{refl}_{r^i}, \lambda s.\mathsf{refl}_{e^i(s)}) \tag{60}$$

Now, we can generalize the situation: we claim that, *for all* $(\psi^1, f^1, \ldots)$, we can derive the induction principle plus two β-equalities. This formulation allows us to use based path induction on (f^1, ψ^1) and assume that $f^1 \equiv \lambda b.\mathsf{id}$, $\psi_1 \equiv \mathsf{refl}_{\lambda b.\mathsf{id}}$. This lets the mentioned dependencies disappear and we get $\psi^2 : \delta^1 = \mathsf{refl}_{r^i}$ as well as $\psi^3 : \gamma^1 = \lambda s.\mathsf{refl}_{e^i(s)}$. In addition, (59) simplifies to $\mathsf{ind}_{r,e}(q) :\equiv \mathsf{proj}_2(f_b(q))$.

For the first β-equality, we unfold the type of δ:

$$\delta : (\mathsf{refl}_{a_0}, \mathsf{ind}_{r,e}(\mathsf{refl}_{a_0})) = (\mathsf{refl}_{a_0}, r) \tag{61}$$

We need to show that the second components are equal. From δ, we get that the second components are equal when one is transported along the δ^1, and from ψ^1, we get that this is a transport along refl.

The procedure for the second β-equation is similar. The details are best seen by considering the following diagram:

$$\begin{tikzcd}
{[a_0] = [b]} \arrow[r, "f_b"] \arrow[d, "\_ \cdot \mathsf{glue}(s)"'] \arrow[dr, phantom, "\gamma"] & \Sigma(q : [a_0] = [b]).P(q) \arrow[d, "\_ \cdot \mathsf{glue}(s), e(\_, s, \_)"] \\
{[a_0] = [c]} \arrow[r, "f_b"'] & \Sigma(q : [a_0] = [c]).P(q)
\end{tikzcd} \tag{62}$$

γ says that this square commutes. Let us take some $q : [a_0] = [b]$ and see how it is mapped (using $f_1 \equiv \mathsf{id}$ and so on):

$$\begin{tikzcd}
q \arrow[r, mapsto] \arrow[dd, mapsto] & (q, \mathsf{ind}_{r,e}(q)) \arrow[d, mapsto] \\
 & (q \cdot \mathsf{glue}(s), e(q, s, \mathsf{ind}_{r,e}(q)) \\
q \cdot \mathsf{glue}(s) \arrow[r, mapsto] & (q \cdot \mathsf{glue}(s), \mathsf{ind}(q \cdot \mathsf{glue}(s)))
\end{tikzcd} \tag{63}$$

Here, γ tells us that the two pairs at the bottom right are equal. As before, we need that their second components are equal; and analogously to what we did before, we use ψ^3 to see that this is the case. □

III. Equality in Pushouts

As discussed in the introduction, pushouts and coequalizers can easily be defined in terms of each other. The standard representation in the HoTT literature as a higher inductive type, where we assume that types L, M, N and functions $f : L \to M$ and $g : L \to N$ are given, is as follows:

$\mathsf{data}\ M \sqcup^L N : \mathcal{U}$

$\quad \mathsf{inl} : M \to M \sqcup^L N$

$\quad \mathsf{inr} : N \to M \sqcup^L N$

$\quad \mathsf{glue} : \Pi(l : L).\mathsf{inl}(f(l)) = \mathsf{inr}(g(l))$

$$\begin{tikzcd}
L \arrow[r, "g"] \arrow[d, "f"'] & N \arrow[d, dashed, "\mathsf{inr}"] \\
M \arrow[r, dashed, "\mathsf{inl}"'] & M \sqcup^L N
\end{tikzcd}$$

We write $\mathsf{in} : (M + N) \to M \sqcup^L N$ for the map given by $(\mathsf{inl}, \mathsf{inr})$. To simplify notation, we keep the inclusions $i_1 : M \to (M + N)$ and $i_2 : N \to (M + N)$ implicit.

Since pushouts are used a lot and play a vital role in the Seifert-van Kampen theorem (cf. Section V), we want to state our main result explicitly for pushouts instead of coequalizers. The proofs can straightforwardly be obtained by expressing the pushouts as coequalizers, as described in the introduction.[1]

Theorem 14 (induction for pushout equality)**.** *Assume L, M, N, f, g are given as above, together with a point $n_0 : N$. Assume we are given families P, Q and terms r, e as follows:*

$$P : \Pi\{m : M\}.(\mathsf{inr}(n_0) = \mathsf{inl}(m)) \to \mathcal{U} \tag{64}$$
$$Q : \Pi\{n : N\}.(\mathsf{inr}(n_0) = \mathsf{inr}(n)) \to \mathcal{U} \tag{65}$$
$$r : Q(\mathsf{refl}_{\mathsf{inr}(n_0)}) \tag{66}$$
$$e : \Pi(l : L), (q : \mathsf{inr}(n_0) = \mathsf{inl}(f(l))). \quad P(q) \simeq Q(q \cdot \mathsf{glue}(l)). \tag{67}$$

Then, we can construct terms

$$\mathsf{ind}^P_{r,e} : \Pi\{m : M\}, (q : \mathsf{inr}(n_0) = \mathsf{inl}(m)).P(q) \tag{68}$$
$$\mathsf{ind}^Q_{r,e} : \Pi\{n : N\}, (q : \mathsf{inr}(n_0) = \mathsf{inr}(n)).Q(q) \tag{69}$$

with the following β-rules:

$$\mathsf{ind}^P_{r,e}(\mathsf{refl}_{\mathsf{inr}(n_0)}) = r \tag{70}$$
$$\mathsf{ind}^Q_{r,e}(q \cdot \mathsf{glue}(l)) = e(l, q, \mathsf{ind}^P_{r,e}(q)) \tag{71}$$

□

Remark 15. As before, the first β-rule (70) holds judgmentally in our formalization.

Theorem 16 (initiality of pushout equality)**.** *Given the same data as in the previous theorem, we can consider the category $\mathcal{P}$, whose definition mirrors that of $\mathcal{C}$. Objects are quadruples (J, K, r, e),*

$$J : M \to \mathcal{U} \quad (72) \qquad r : K(n_0) \tag{74}$$
$$K : N \to \mathcal{U} \quad (73) \qquad e : \Pi(l : L).J(f(l)) \simeq K(g(l)) \tag{75}$$

and a morphism between (J, K, r, e) and (J', K', r', e') consists of fiberwise functions which preserve r and commute with e.

Then, the object defined by

$$J^i(m) :\equiv (\mathsf{inr}(n_0) = \mathsf{inl}(m)) \tag{76}$$
$$K^i(n) :\equiv (\mathsf{inr}(n_0) = \mathsf{inr}(n)) \tag{77}$$
$$r :\equiv \mathsf{refl}_{\mathsf{inr}(n_0)} \tag{78}$$
$$e(l) :\equiv \_ \cdot \mathsf{glue}(l) \tag{79}$$

is initial in $\mathcal{P}$. □

IV. First Applications

We anticipate that our main result, especially in the formulations of Theorem 3 and 16, will be a useful tool for a variety of constructions in HoTT. Our own motivation for developing these theorems is the concrete realization of the plans outlined by the first-named author [24]. In this paper, we present two shorter applications.

[1]In Lean, this is simply a specialization.

A. The Loop Space of the Circle

Recall that the *loop space* $\Omega(X)$ of a type X with an (implicitly given) point $x_0 : X$ is defined to be $x_0 = x_0$. Thus, the loop space of the circle $\mathbb{S}^1$ (4) is simply $\mathsf{base} = \mathsf{base}$. Let us reprove the following known result:

Theorem 17 (Licata-Shulman [1])**.** $\Omega(\mathbb{S}^1) \simeq \mathbb{Z}$.

Proof. As discussed in the introduction, $\mathbb{S}^1$ is the coequalizer of $\mathbf{1}$ and the relation which has $\mathbf{1}$ as its value. This allows us to apply Theorem 7 and, since all quantifications are now quantifications over the unit type, we can safely ignore them. Thus, $\big(\Omega(\mathbb{S}^1), \mathsf{refl}, \_ \cdot \mathsf{loop}\big)$ is the initial object in the category of pointed types with an automorphism. Due to the uniqueness of initial objects, all we need is that $(\mathbb{Z}, 0, \mathsf{suc})$ is initial in this category. This statement is completely removed from the higher inductive type $\mathbb{S}^1$; it is a basic property of the integers, analogous to the fact that $(\mathbb{N}, 0, \mathsf{suc})$ is initial in the category of pointed types with an endofunction. $\square$

Of course, the difficulty of a concrete proof for the initiality property depends on the concrete definition of $\mathbb{Z}$ that one uses. With the definition used by Licata and Shulman (essentially $\mathbb{N} + \mathbf{1} + \mathbb{N}$), this is easy albeit some work. We will come back to definitions of the integers in Remark 21.

B. Pushouts Preserve Embeddings

Recall that an *embedding* is a map $h : X \to Y$ whose fibers are propositions, i.e. where, for each $y : Y$, the type $h^{-1}(y) :\equiv \Sigma(x : X).y = h(x)$ is a ("mere") proposition. Equivalently, h is an embedding if and only if

$$\mathsf{ap}_h : \Pi\{x, x' : X\}.(x = x') \to (h(x) = h(x')) \qquad (80)$$

is a family of equivalences between path spaces. As formalized by Finster [18] via an encode-decode construction, embeddings are closed under pushout. As a further application of our main result, we present an alternative (and significantly shorter) argument.

Theorem 18 (Finster [18])**.** *Embeddings are closed under pushout. Explicitly, if f in the diagram on the right is an embedding, then so is* inr.

$$\begin{tikzcd} L \arrow[r, "g"] \arrow[d, hook, "f"'] & N \arrow[d, hook, dashed, "\mathsf{inr}"] \\ M \arrow[r, dashed, "\mathsf{inl}"'] & M \sqcup^L N \end{tikzcd}$$

Proof. Using (80), we need to show that $\mathsf{ap}_{\mathsf{inr}} : (n_0 = n) \to (\mathsf{inr}(n_0) = \mathsf{inr}(n))$ is an equivalence for all points n_0, n. Thus, for any $q : \mathsf{inr}(n_0) = \mathsf{inr}(n)$, we want to find something in the fiber over q. This tells us how we need to choose the type family Q (65) of Theorem 14: we fix n_0 and define

$$Q : \Pi(n : N).(\mathsf{inr}(n_0) = \mathsf{inr}(n)) \to \mathcal{U} \qquad (81)$$

$$Q(n, q) :\equiv \mathsf{ap}_{\mathsf{inr}}^{-1}(q). \qquad (82)$$

We also need to define the type family P (64). Given something in M, we "move" it back to N by going via the fiber, which allows us to define P using Q:

$$P : \Pi(m : M).(\mathsf{inr}(n_0) = \mathsf{inl}(m)) \to \mathcal{U} \qquad (83)$$

$$P(m, q) :\equiv \Sigma((l_0, q_0) : f^{-1}(m)). \\ Q\big(g(l_0), q \cdot \mathsf{ap}_{\mathsf{inl}}(q_0) \cdot \mathsf{glue}(l_0)\big). \qquad (84)$$

The component r (66) is the obvious one, $r :\equiv (\mathsf{refl}, \mathsf{refl})$. For a given $l : L$ we know that, since f is an embedding, the type $f^{-1}(f(l))$ is contractible and we can assume $(l_0, q_0) \equiv (l, \mathsf{refl})$. This implies $P(f(l), q) \simeq Q(g(l), q \cdot \mathsf{glue}(l))$, which is exactly what we need in order to define the component e (67). Thus, we have

$$\mathsf{ind}^Q_{r,e} : \Pi\{n : N\}, (q : \mathsf{inr}(n_0) = \mathsf{inr}(n)).\mathsf{ap}_{\mathsf{inr}}^{-1}(q), \qquad (85)$$

i.e. a section s of $\mathsf{ap}_{\mathsf{inr}}$ (a function such that $\mathsf{ap}_{\mathsf{inr}} \circ s = \mathsf{id}$). To show that $s \circ \mathsf{ap}_{\mathsf{inr}} : (n_0 = n) \to (n_0 = n)$ is the identity, we do path induction and use the first β-rule (70). $\square$

V. Free Groupoids and a Higher Seifert-van Kampen Theorem

The traditional Seifert-van Kampen (SvK) theorem, a standard result in algebraic topology, makes it possible to calculate the fundamental group of a topological space X when the fundamental groups of two open and path-connected subspaces covering X are already known. Favonia and Shulman [2] have stated and shown this theorem in HoTT, where the union of subspaces can be phrased as a (homotopy) pushout. Their result is that fundamental groups of a pushout are equivalent to a type code which they define as a set-quotient of a list.

Fundamental groups (in topology) are quotients of spaces or (in HoTT) are 0-truncations of equality types. Thus, it is natural to ask for a *higher dimensional* version of the theorem which does not quotient or truncate. In homotopy theory, different versions have been proved by Lurie [25] and Brown, Higgins, and Sivera [26]. In HoTT, it is an open problem how this could be done. Our results of the current paper suggest one possible such higher SvK theorem, which (after recalling the Favonia-Shulman result) we present in this section.

Note that the precise formulation of a theorem is part of the open question how to generalize the SvK theorem in HoTT, since the analogue of the code family by Favonia and Shulman has to be defined (and a trivial solution exists: define this analogue to be the equality). Our justification for why the analogue we suggest is reasonable is that, by 0-truncating, the Favonia-Shulman theorem can be recovered relatively easily. As before in Section III, let us assume that the types L, M, N and functions f, g in the pushout on the right are given for the rest of the section. As in [2], we write $P :\equiv M \sqcup^L N$.

$$\begin{tikzcd} L \arrow[r, "g"] \arrow[d, "f"'] & N \arrow[d, dashed, "\mathsf{inr}"] \\ M \arrow[r, dashed, "\mathsf{inl}"'] & P \end{tikzcd}$$

A caveat is in order. In this section, we make use of *indexed* higher inductive types, and this is not part of our formalization. Note that indexed inductive types can always be encoded via inductive types [27], [28], and we expect that the same is true for indexed higher inductive types.

A. The Favonia-Shulman SvK Theorem

Favonia and Shulman give two versions of the SvK theorem. We concentrate on the first ("naive Seifert-van Kampen Theorem"); we think the difference between the two versions is not really relevant for what we present in the current paper. We do not repeat their definition of code in full detail, since this definition is of significant length (2 pages including careful explanations and remarks). In a nutshell, $\mathsf{code}(u, v)$ is a set-quotient of a type of lists which "link" u and v, where $u, v : P$. For simplicity, we restrict ourselves to endpoints in $M + N$ (instead of P). Let us fix $n_0, n : N$. Then, the considered lists are points $k_1, l_1, k_2, l_2, \ldots : L$ together with $p_i : \|g(l_i) = g(k_{i+1})\|_0$ and $q_i : \|f(k_i) = f(l_i)\|_0$ such that the p_i and q_i form a path from n_0 to n as in the following drawing, where the vertical arrows are glue's:

$$\begin{tikzcd}
n_0 \arrow[r, "p_0"] & g(k_1) \arrow[d] & g(l_1) \arrow[r, "p_1"] & g(k_2) \arrow[d] & g(l_2) \arrow[r, "p_2"] & n \\
& f(k_1) \arrow[r, "q_1"] & f(l_1) \arrow[u] & f(k_2) \arrow[r, "q_2"] & f(l_2) \arrow[u] &
\end{tikzcd} \tag{86}$$

Next, a set-quotient is taken which ensures that we can remove "trivial" paths in the above picture. For example, if $l_1 \equiv k_2$ and $p_1 \equiv \mathsf{refl}_{f(l_1)}$, then the set-quotient ensures that the above list is identified with the following:

$$\begin{tikzcd}
n_0 \arrow[r, "p_0"] & g(k_1) \arrow[d] & g(l_2) \arrow[r, "p_2"] & n \\
& f(k_1) \arrow[r, "q_1 \cdot q_2"] & f(l_2) \arrow[u] &
\end{tikzcd} \tag{87}$$

This set-quotient defines the type $\mathsf{code}(\mathsf{inr}(n_0), \mathsf{inr}(n))$, and the definition where one or both endpoints are in M is analogous. Restricted to the case where we consider endpoints in $M + N$, the SvK theorem states:

Theorem 19 (Favonia and Shulman [2]). *For $x, y : M + N$, there is an equivalence $\|\mathsf{in}(x) =_P \mathsf{in}(y)\|_0 \simeq \mathsf{code}(x, y)$.* □

B. From Quotiented Lists to Free Higher Groupoids

The central difficulty of a higher version of the SvK theorem is, of course, avoiding the set-truncation. Note that, in the above description of the lists, the set-truncations in $p_i : \|g(l_i) = g(k_{i+1})\|_0$ and $q_i : \|f(k_i) = f(l_i)\|_0$ can be removed since we set-truncate later when taking the set-quotient. This is essentially a repeated application of the equivalence

$$\|\Sigma(a : A). \|B(a)\|_n\|_n \simeq \|\Sigma(a : A). B(a)\|_n. \tag{88}$$

This unnecessary set-truncation *does* make sense in the formulation of the SvK theorem, where all equality types are set-truncated, but removing it makes it easier to motivate our *higher* SvK theorem.

Next, we suggest an alternative definition for the type of lists (before quotienting/truncation). To simplify things further, let us fix $n_0 : N$ and consider lists starting at this point. Let us now look at the following *indexed* inductive type $C_0 : (M+N) \to \mathcal{U}$ with three constructors, where $C_0(x)$ should be understood as a type of lists from n_0 to x. Recall that we keep the embeddings $i_1 : M \to (M + N)$ and $i_2 : N \to (M + N)$ implicit.

$$\begin{aligned}
&\mathsf{data}\ C_0 : (M + N) \to \mathcal{U} \\
&\quad \mathsf{nil} : C_0(n_0) \\
&\quad \mathsf{gl} : \Pi(l : L). C_0(f(l)) \to C_0(g(l)) \\
&\quad \mathsf{gl}' : \Pi(l : L). C_0(g(l)) \to C_0(f(l))
\end{aligned} \tag{89}$$

Clearly, nil gives us the empty list. The other two constructors allow us to switch between lists ending in a point in M to lists ending in a point in N and vice versa. Intuitively, this is done simply by adding a glue at the end of the list. This explains how to add the *vertical* lines of a list as drawn in (86). It may be surprising that we do not add the *horizontal* components p_i and q_i explicitly. The reason is that they are automatically and implicitly present in this encoding: the map $\mathsf{transport}^{C_0}$ of type

$$\Pi\{l, l' : L\}. (g(l) = g(l')) \to \big(C_0(g(l)) \to C_0(g(l'))\big) \tag{90}$$

allows us to "insert" the upper horizontal components in (86) and (exchanging g by f) also the lower horizontal components.

The type $C_0(x)$ encodes lists from n_0 to x, but we have not done the quotienting, i.e. the lists (86) and (87) are still different. To remedy this, we can turn C_0 into an indexed *higher* inductive type and add constructors ensuring that $\mathsf{gl}(l, \mathsf{gl}'(l, x)) = x$ and $\mathsf{gl}'(l, \mathsf{gl}(l, x)) = x$. If we set-truncate, this would give us the correct type, namely something equivalent to the $\mathsf{code}(n_0, x)$ by Favonia and Shulman. Since we do not want to set-truncate, we have to be more careful. $\mathsf{gl}(l)$ and $\mathsf{gl}'(l)$ together with the equality constructors will form a pair of *quasi-inverses* (cf. [3]), and it is known that this type is not well-behaved. Instead, we mirror the components that form an actual equivalence. Although there are several formulations that would work, we use those that turn gl into a *bi-invertible map* [3], as follows:

$$\begin{aligned}
&\mathsf{data}\ C : (M + N) \to \mathcal{U} \\
&\quad \mathsf{nil} : C(n_0) \\
&\quad \mathsf{gl} : \Pi(l : L). C(f(l)) \to C(g(l)) \\
&\quad \mathsf{linv} : \Pi(l : L). C(g(l)) \to C(f(l)) \\
&\quad \mathsf{leq} : \Pi(l : L), (x : C(f(l))). \mathsf{linv}(l, \mathsf{gl}(l, x)) = x \\
&\quad \mathsf{rinv} : \Pi(l : L). C(g(l)) \to C(f(l)) \\
&\quad \mathsf{leq} : \Pi(l : L), (y : C(g(l))). \mathsf{gl}(l, \mathsf{rinv}(l, y)) = y
\end{aligned} \tag{91}$$

This definition of C does certainly not look very appealing, and we only give this presentation because it is the "standard" way of presenting higher inductive types. If we allow ourselves to fold the last five constructors into a single one, the type looks as follows:

$$\begin{aligned}
&\mathsf{data}\ C : (M + N) \to \mathcal{U} \\
&\quad \mathsf{nil} : C(n_0) \\
&\quad \mathsf{gl} : \Pi(l : L). C(f(l)) \simeq C(g(l))
\end{aligned} \tag{92}$$

It may also be interesting to do this in the formulation for a coequalizer instead of a pushout. As explained in Section I-B, this is a completely mechanical translation. Thus, assume A with $a_0 : A$ and $\sim$. Then, the corresponding type G in the "folded" form looks as follows:

$$\begin{aligned}
&\mathsf{data}\ G : A \to \mathcal{U} \\
&\quad \mathsf{nil} : G(a_0) \\
&\quad \mathsf{cons} : \Pi\{b, c : A\}. (b \sim c) \to G(b) \simeq G(c)
\end{aligned} \tag{93}$$

Let us write $\mathfrak{G}(a_0, \_)$ instead of $G(\_)$, in order to explicitly mention the point a_0. We can call $\mathfrak{G}$ the *free higher groupoid* generated by $\sim$. This construction generalizes the explicit construction of a free higher group (based on an idea by Capriotti, cf. [29]). It also generalizes the "integer type as a higher inductive type" (itself a special case of the free higher group) which was independently suggested by Pinyo and Altenkirch [30] (based on Capriotti's idea), by van der Weide et al. in unpublished work, and in a formalization by Cavallo based on a remark by Mörtberg [31]. This example is discussed further in Remark 21 below.

C. A Higher SvK Theorem

The type family C depends on the chosen point n_0. To remove this dependency, let us consider a version of C which is indexed twice over $(M+N)$: we write $\mathfrak{C}(n_0, y)$ for $C(y)$. This expression plays the role of code in our higher analogue of the Favonia-Shulman result, Theorem 19. While it can be extended to a family $P \to P \to \mathcal{U}$ in a straightforward way, we choose the following formulation for simplicity (and to match Theorem 19 more closely):

Theorem 20 (a higher Seifert-van Kampen theorem)**.** *For $x, y : M+N$, we have an equivalence:*

$$(\mathsf{in}(x) =_P \mathsf{in}(y)) \simeq \mathfrak{C}(x, y). \tag{94}$$

Proof. Like all (indexed/higher/ordinary) inductive types, (92) is (homotopy-) initial in an appropriately formulated category of algebras (see [19], [32], and others). Here is where we draw the connection with the main result of the paper: The category in which (92) is initial is the category $\mathcal{P}$ from Theorem 16.[2] This is easy to see when we use the general specification and definition of higher inductive-inductive types given by Kaposi and Kovács [33], [34], but see Remark 21 below.

By the uniqueness of the initial object and by Theorem 16, $C(x)$ is equivalent to $\mathsf{inr}(n_0) =_P \mathsf{in}(x)$. Letting n_0 vary, we get the statement of the theorem. □

It is relatively straightforward to recover the set-truncated SvK statement (Theorem 19) from the higher version (Theorem 20). We can simply set-truncate both sides in (94) and then prove that $\|\mathfrak{C}(x,y)\|_0$ is equivalent to $\mathsf{code}(x, y)$ by constructing maps in both directions.

Remark 21. Theorem 20 and its proof deserve additional comments. We think it is fair to say that the formal theory of indexed higher inductive types is not yet well-established, but it is under very active development. Kaposi and Kovács ([33], [34]) have suggested a definition for general higher inductive-inductive types which captures the case we need. Indexed higher inductive types are considered in some of the cubical settings; cf. Cavallo and Harper [35], and there are plans to extend cubical Agda [36], [37], [38] and redtt [39] with the concept (at the time of writing, a possibly not final version is available in cubical Agda). The syntax in (91) is the obvious and non-controversial one for such indexed higher inductive types. We think it would be desirable to also allow the syntactical representation in (92), even if only as syntactic sugar for (91). Note that Kaposi and Kovács allow equalities between types, which is very similar to allowing this family of equivalences.

The critical step in the above proof of Theorem 20 is to establish (92) as the initial object of the category $\mathcal{P}$. With the specification suggested by Kaposi and Kovács allowing (92), with equalities instead of equivalences, this part is easy. However, we want to emphasize that the initiality of (91) is not immediate at all if we use what we could call the *direct induction principle*[3]. The direct induction principle is the "standard" principle one derives by giving one case for each constructor, as done in the book [3] and by current proof assistants such as cubical Agda. Unfortunately, due to the type dependency in the direct induction principle, it becomes very hard to "fold" the components for the type (91) in order to achieve the principle one would expect from (92). We expect that implementing Theorem 20 in cubical Agda would be extremely tedious for this reason.

The core of the problem with the direct induction principle is that it does not allow us to "reason on the level of constructors". As an example, let us consider the interval with two point constructors and one path constructor. If we can reason on the level of constructors, it is by "singleton contraction" clear that one point and the path constructor form a contractible pair, and that the interval is therefore equivalent to the type generated by a single point. With the direct induction principle, this style of reasoning is not possible. It turns out to be easy enough to prove the interval contractible, but in other cases, the situation is less fortunate.

As an example, proposals by Pinyo and Altenkirch [30], unpublished work by van der Weide et al., and a formalization by Cavallo based on a remark by Mörtberg [31] suggest to define $\mathbb{Z}$ as a higher inductive type, and their very definition is chosen such that $\mathbb{Z}$ should become the initial object of the category of pointed types with automorphism (cf. Section IV-A). Their definitions are versions of (93) with A and $\sim$ replaced by the unit type and the relation constantly unit. Crucially, they have to "unfold" the constructor cons, since this is what the current cubical proof assistants require. It turns out that this makes it extremely tedious to prove the resulting type equivalent to other definitions of the integers.

VI. Final Remarks

We have shown a theorem, reminiscent of an induction principle, which allows to reason about path spaces of pushouts/coequalizers. There are multiple reasonable formulations of this result. We have then proceeded to use this result for short proofs of two statements that had formerly been proved with encode-decode constructions.

Kristina Sojakova has formulated an alternative version of the proof of Theorem 3. This proof is presented in a more direct fashion, without explicitly going through initiality in wild categories, although all analogous steps are still taken.

[2]To be precise, the object $(C \circ i_1, C \circ i_2, \mathsf{nil}, \mathsf{gl})$ is initial in $\mathcal{P}$.

[3]The terminology was suggested by Anders Mörtberg in a discussion with the authors.

Such a presentation makes it easier to see that the first β-rule in Theorem 3 and Theorem 14 holds judgmentally.

The core of the proof in Section II is the isomorphism between $\mathcal{C}$ and $\mathcal{D}$ (Lemma 11). Strictly speaking, the full isomorphism is not required since we are only interested in the initial objects, but showing the isomorphism seems conceptually cleaner and is not significantly harder that a more minimalistic approach.

A question to consider in the future would be whether it is possible to generalize the result from coequalizers to arbitrary higher inductive types or at least to a larger fragment of higher inductive types.

Acknowledgments

We would like to thank Kristina Sojakova, Paolo Capriotti, Anders Mörtberg, and Michael Shulman for their comments, for valuable suggestions, and for inspiring discussions. We are also grateful to the anonymous reviewers for their remarks, which have helped us to improve this paper.

References

[1] D. Licata and M. Shulman, “Calculating the fundamental group of the circle in homotopy type theory,” in *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2013, pp. 223–232.

[2] K.-B. Hou (Favonia) and M. Shulman, “The Seifert-van Kampen theorem in homotopy type theory,” in *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 62. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, pp. 22:1–22:16. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2016/6562

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

[4] F. van Doorn, “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*, 2016, pp. 122–129. [Online]. Available: http://doi.acm.org/10.1145/2854065.2854076

[5] N. Kraus, “Constructions with non-recursive higher inductive types,” in *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. New York, NY, USA: ACM, 2016, pp. 595–604. [Online]. Available: http://doi.acm.org/10.1145/2933575.2933586

[6] E. Rijke, “The join construction,” 2017, arXiv:1701.07538.

[7] L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer, “The lean theorem prover,” in *Automated Deduction - CADE-25, 25th International Conference on Automated Deduction*, 2015.

[8] F. van Doorn, J. von Raumer, and U. Buchholtz, “Homotopy type theory in lean,” in *Interactive Theorem Proving*. Cham: Springer International Publishing, 2017, pp. 479–495.

[9] S. Boulier, E. Rijke, and N. Tabareau, “A coinductive approach to type valued equivalence relations,” 2017, abstract presented at the workshop on HoTT/UF in Oxford.

[10] U. Buchholtz, F. van Doorn, and E. Rijke, “Higher groups in homotopy type theory,” in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, ser. LICS ’18. New York, NY, USA: ACM, 2018, pp. 205–214. [Online]. Available: http://doi.acm.org/10.1145/3209108.3209150

[11] U. Buchholtz and K.-B. H. (Favonia), “Cellular cohomology in homotopy type theory,” in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, 2018.

[12] U. Buchholtz and E. Rijke, “The Cayley-Dickson construction in homotopy type theory,” *arXiv preprint arXiv:1610.01134*, 2016.

[13] ——, “The real projective spaces in homotopy type theory,” in *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, 06 2017, pp. 1–8.

[14] D. Licata and E. Finster, “Eilenberg-MacLane spaces in homotopy type theory,” in *Proceedings of the 29th Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, 2014, pp. 66–74.

[15] D. Licata and G. Brunerie, “$\pi_n(S^n)$ in homotopy type theory,” ser. LNCS, vol. 8307. Springer, 2013, pp. 1–16.

[16] G. Brunerie, “The James construction and $\pi_4(\mathrm{s}^3)$ in homotopy type theory,” *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1710.10307

[17] C. Paulin-Mohring, “Inductive definitions in the system Coq - rules and properties,” in *Typed Lambda Calculi and Applications (TLCA)*, ser. Lecture Notes in Computer Science, M. Bezem and J. F. Groote, Eds., no. 664, 1993.

[18] E. Finster, “Agda library file pushoutmono,” 2017, available on GitHub at https://github.com/HoTT/HoTT-Agda/blob/master/theorems/stash/modalities/gbm/PushoutMono.agda.

[19] S. Awodey, N. Gambino, and K. Sojakova, “Homotopy-initial algebras in type theory,” *J. ACM*, vol. 63, no. 6, pp. 51:1–51:45, Jan. 2017. [Online]. Available: http://doi.acm.org/10.1145/3006383

[20] K. Sojakova, “Higher inductive types as homotopy-initial algebras,” in *Principles of Programming Languages (POPL)*. New York, NY, USA: ACM, 2015, pp. 31–42. [Online]. Available: http://doi.acm.org/10.1145/2676726.2676983

[21] P. Capriotti and N. Kraus, “Univalent higher categories via complete semi-segal types,” *Proc. ACM Program. Lang.*, vol. 2, no. POPL, pp. 44:1–44:29, Dec. 2017. [Online]. Available: http://doi.acm.org/10.1145/3158132

[22] B. Ahrens, K. Kapulkin, and M. Shulman, “Univalent categories and the Rezk completion,” *Mathematical Structures in Computer Science (MSCS)*, pp. 1–30, Jan 2015. [Online]. Available: http://journals.cambridge.org/article_S0960129514000486

[23] F. van Doorn and U. Buchholtz, “The dependent universal property,” 2017, lean library file, available on GitHub at https://github.com/gebner/hott3/.

[24] N. Kraus, “Some connections between open problems,” 2018, Homotopy Type Theory Electronic Seminar Talks, hosted by Dan Christensen and Chris Kapulkin. Video and slides available on the website.

[25] J. Lurie, “Derived algebraic geometry vi: Ek algebras,” *arXiv preprint arXiv:0911.0018*, 2009.

[26] R. Brown, P. J. Higgins, and R. Sivera, *Nonabelian algebraic topology*, 2011.

[27] T. Altenkirch, N. Ghani, P. Hancock, C. McBride, and P. Morris, “Indexed containers,” *J. Funct. Program.*, vol. 25, 2015. [Online]. Available: https://doi.org/10.1017/S095679681500009X

[28] C. Sattler, “On relating indexed w-types with ordinary ones,” 2015, abstract, presented at TYPES’15.

[29] N. Kraus and T. Altenkirch, “Free higher groups in homotopy type theory,” in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, ser. LICS ’18. New York, NY, USA: ACM, 2018, pp. 599–608. [Online]. Available: http://doi.acm.org/10.1145/3209108.3209183

[30] G. Pinyo and T. Altenkirch, “Integers as a higher inductive type,” 2018, abstract, presented at TYPES’18.

[31] E. Cavallo and A. Mörtberg, “Successor on biinv-int which cancels pred exactly,” Dec 2018, redtt implementation, available online at https://github.com/RedPRL/redtt/blob/master/library/cool/biinv-int.red.

[32] T. Coquand, S. Huber, and A. Mörtberg, “On higher inductive types in cubical type theory,” in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, ser. LICS ’18. New York, NY, USA: ACM, 2018, pp. 255–264. [Online]. Available: http://doi.acm.org/10.1145/3209108.3209197

[33] A. Kaposi and A. Kovács, “A syntax for higher inductive-inductive types,” in *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 108. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 20:1–20:18. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2018/9190

[34] ——, “Signatures and induction principles for higher inductive-inductive types,” *arXiv preprint arXiv:1902.00297*, 2019.

[35] E. Cavallo and R. Harper, “Higher inductive types in cubical computational type theory,” *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, p. 1, 2019.

[36] A. Vezzosi, “Cubical Agda,” 2018, extension to Agda, available in the main Agda repository at https://github.com/agda/agda.

[37] A. Mörtberg, “Cubical agda,” 2018, blog post at https://homotopytypetheory.org/2018/12/06/cubical-agda/.

[38] A. Mörtberg and A. Vezzosi, “An experimental library for cubical agda,” 2018, online at https://github.com/agda/cubical.

[39] C. Angiuli, E. Cavallo, K.-B. H. (Favonia), R. Harper, A. Mörtberg, and J. Sterling, “redtt – cartesian cubical proof assistant,” 2018, talk available online at http://www.jonmsterling.com/pdfs/dagstuhl.pdf, implementation at https://github.com/RedPRL/redtt.