Coherence via Well-Foundedness

Taming Set-Quotients in Homotopy Type Theory

Nicolai Kraus
University of Nottingham
University of Birmingham
United Kingdom

Jakob von Raumer
University of Nottingham
United Kingdom

Abstract

Suppose we are given a graph and want to show a property for all its *cycles* (closed chains). Induction on the length of cycles does not work since sub-chains of a cycle are not necessarily closed. This paper derives a principle reminiscent of induction for cycles for the case that the graph is given as the symmetric closure of a locally confluent and (co-)well-founded relation. We show that, assuming the property in question is sufficiently nice, it is enough to prove it for the empty cycle and for cycles given by local confluence.

Our motivation and application is in the field of *homotopy type theory*, which allows us to work with the higher-dimensional structures that appear in homotopy theory and in higher category theory, making *coherence* a central issue. This is in particular true for quotienting – a natural operation which gives a new type for any binary relation on a type and, in order to be well-behaved, cuts off higher structure (*set-truncates*). The latter makes it hard to characterise the type of maps from a quotient into a higher type, and several open problems stem from this difficulty.

We prove our theorem on cycles in a type-theoretic setting and use it to show coherence conditions necessary to eliminate from set-quotients into 1-types, deriving approximations to open problems on free groups and pushouts. We have formalised the main result in the proof assistant Lean.

CCS Concepts: • **Theory of computation** → *Constructive mathematics*; *Logic and verification*; **Type theory**; *Equational logic and rewriting*; • **Mathematics of computing** → *Graphs and surfaces*.

1 Introduction

Homotopy type theory [27], commonly known as *HoTT*, is a version of constructive dependent type theory inspired by the observation that types carry the structure of (higher) groupoids [13, 23, 28] and allow homotopical interpretations [4, 19]. The assumption of *uniqueness of identity/equality proofs* (UIP), which says that every equality type (written as $a = b$) has at most one element, is rejected; instead, types satisfying this principle are known as *sets* (or *homotopy sets, h-sets*).

This leads to new opportunities. We can find "higher" types (i.e. "non-sets"), whose equality structure may be very complicated. At the same time, the familiar and desirable concept of quotienting [12] (from now on called *set-quotienting* [27, Chp 6.10]) becomes an instance of the more general idea of *higher inductive types*. With the help of these, theories of higher structures can be developed synthetically in type theory. One difficulty of constructions is that it is usually not sufficient to know that two things are equal, but we need to know *how* they are equal. Families of equalities need to "match" or "fit together": they need to be *coherent*.

The guiding motivation for the theory that we develop in this paper is the needed coherence that arises when working with set-quotients simultaneously with higher types. Recall from [27, Chp 6.10] that, for a given type $A : \mathcal{U}$ and a relation[1] $(\sim) : A \to A \to \mathcal{U}$, the set-quotient can be

---

[1] A *relation* is simply a type family $A \to A \to \mathcal{U}$ of which we do not assume any properties. In particular, $\sim$ does not need to be an equivalence relation for the quotient to exist. Some authors call such a relation *proof-relevant* to emphasise that it is not necessarily valued in propositions.

implemented as the higher inductive type

$$\text{inductive } A/\sim \text{ where}$$
$$\iota : A \to A/\sim$$
$$\text{glue} : \Pi\{a, b : A\}.(a \sim b) \to \iota(a) = \iota(b) \quad (1)$$
$$\text{trunc} : \Pi\{x, y : A/\sim\}.\Pi(p, q : x = y).p = q$$

From this representation, we can derive the elimination rule: In order to get a function $f : (A/\sim) \to X$, we need to give a function $g : A \to X$ such that, whenever $a \sim b$, we have $g(a) = g(b)$. However, this only works if X is a set itself. If it is not, we have a priori no way of constructing the function f.

Let us look at one instance of the problem. We consider the following set-quotient, which we will use as a running example. It is a standard construction that has been discussed in [27, Chp 6.11].

Example 1 (free group). Let A be a set. We construct the free group on A as a set-quotient. We consider lists over $A + A$, where we think of the left copy of A as positive and the right copy as negative elements. For $x : A + A$, we write x^{-1} for the "inverted" element:

$$\text{inl}(a)^{-1} :\equiv \text{inr}(a) \qquad\qquad \text{inr}(a)^{-1} :\equiv \text{inl}(a) \quad (2)$$

The binary relation $\sim$ on $\text{List}(A + A)$ is generated by

$$[\ldots, x_1, x_2, x_2^{-1}, x_3, \ldots] \;\sim\; [\ldots, x_1, x_3, \ldots]. \quad (3)$$

Then, the set-quotient $\text{List}(A + A)/\sim$ is the free group on A: It satisfies the correct universal property by [27, Thm 6.11.7].

Another way to construct the free group on A is to re-use the natural groupoid structure that every type carries; this can be seen as a typical "HoTT-style" construction. It works as follows. The *wedge of A-many circles* is the (homotopy) coequaliser of two copies of the map A into the unit type, $\text{hcolim}(A \rightrightarrows \mathbf{1})$. Using a higher inductive type, it can be explicitly constructed:

$$\text{inductive } \text{hcolim}(A \rightrightarrows \mathbf{1}) : \mathcal{U} \text{ where}$$
$$\text{base} : \text{hcolim}(A \rightrightarrows \mathbf{1}) \quad (4)$$
$$\text{loop} : A \to \text{base} = \text{base}$$

Its loop space $\Omega(\text{hcolim}(A \rightrightarrows \mathbf{1}))$ is by definition simply $\text{base} = \text{base}$. This loop space carries the structure of a group in the obvious way: the neutral element is given by reflexivity, multiplication is given by path composition, symmetry by path reversal, and every $a : A$ gives rise to a group element $\text{loop}(a)$. This construction is completely reasonable without the assumption that A is a set, and it therefore defines the free *higher* group (cf. [20]); from now on, we write F_A for it:

$$F_A :\equiv \Omega(\text{hcolim}(A \rightrightarrows \mathbf{1})). \quad (5)$$

In contrast to this observation, the set-quotient of Example 1 ignores any existing higher structure (cf. [27, Rem 6.11.8]) and thus really only defines the free "ordinary" group. If we do start with a set A, it is a natural question whether the

free higher group and the free group coincide: There is a canonical function

$$F_A \to (\text{List}(A + A)/\sim), \quad (6)$$

defined analogously to $\Omega(S^1) \to \mathbb{Z}$, cf. [27]. Classically, this function is an equivalence. Constructively, it is an open problem to construct an inverse of (6).

The difficulties do not stem from the first two constructors of the set-quotient. Indeed, we have a canonical map

$$\omega_1 : \text{List}(A + A) \to F_A \quad (7)$$

which maps a list such as $[\text{inl}(a_1), \text{inr}(a_2), \text{inl}(a_3)]$ to the path composition $\text{loop}(a_1) \cdot (\text{loop}(a_2))^{-1} \cdot \text{loop}(a_3)$. For this map, we also have

$$\omega_2 : \Pi(\ell_1, \ell_2 : \text{List}(A + A)).(\ell_1 \sim \ell_2) \to \omega_1(\ell_1) = \omega_1(\ell_2) \quad (8)$$

since consecutive inverse loops cancel each other out. Therefore, if we define $(A/\!\!/\sim)$ to be the higher inductive type (1) *without* the constructor trunc, i.e. the *untruncated quotient* or *coequaliser*, then there is a canonical map

$$\omega : (\text{List}(A + A)/\!\!/\sim) \to F_A. \quad (9)$$

Thus, the difficulty with defining an inverse of (6) lies solely in the question whether F_A is a set. This is an open problem which has frequently been discussed in the HoTT community (a slight variation is recorded in [27, Ex 8.2]). It is well-known in the community how to circumvent the problem if A has decidable equality. However, the only piece of progress on the general question that we are aware of is the result in [20], where it is shown that all fundamental groups [27, Chp 6.11] are trivial. In other words: Instead of showing that *everything* above truncation level 0 is trivial, the result shows that a single level is trivial. The proof in [20] uses a rather intricate construction which is precisely tailored to the situation.

The construction of functions $(A/\sim) \to X$ is the guiding motivation for the results that we present in this paper. We do not allow an arbitrary type X, however; instead, we assume that X is 1-truncated. As an application, we will give a new proof for the theorem that the fundamental groups of F_A are trivial. We will also show a family of similar statements, by proving a common generalisation.

The characterisation of the equality types of $(A/\sim)$ makes it necessary to consider *cycles*, or *closed zig-zags*, in A. A cycle is simply an element of the symmetric-transitive closure, for example:

Our

$$
\begin{array}{ll}
s : a \sim b & \\
t : c \sim b & \\
p : d \sim c & \\
q : a \sim d &
\end{array}
\qquad
\begin{array}{ccc}
a & \xrightarrow{\;q\;} & d \\
{\scriptstyle s}\downarrow & & \downarrow{\scriptstyle p} \\
b & \xleftarrow[t]{} & c
\end{array}
\quad (10)
$$

first new result (Theorem 13) says: We get a function $(A/\sim) \to X$ into a 1-type X if we have $f : A \to X$ and $h : (a \sim b) \to f(a) = f(b)$, together with the coherence condition stating

that h maps any cycle to a commuting cycle in X. In the case of the example (10) above, this means that the composition $h(s) \cdot h(t)^{-1} \cdot h(p)^{-1} \cdot h(q)^{-1}$ equals $\text{refl}_{f(a)}$. Theorem 13 is fairly simple, and we do not consider it a major contribution of this paper. The actual contribution of the paper is to make Theorem 13 usable since, on its own, it is virtually impossible to apply in any non-trivial situation. The reason for this is that the coherence condition talks about cycles, i.e. *closed* zig-zags. Zig-zags are inductively generated (they are simply a chain of segments), but closed zig-zags are not. If we have a property on cycles, which we cannot generalise to arbitrary zig-zags, then there is no obvious inductive strategy to show the property for all cycles: if we remove a segment of a cycle, the remaining zig-zag is not closed any more. In all our examples, it seems not possible to formulate an induction hypothesis based on not-necessarily-closed zig-zags.

Although the function space $(A/\sim) \to X$ *in homotopy type theory* is the guiding motivation of this paper, the actual main contribution and heart of the paper is completely independent of type theory and could be formulated in almost any (constructive) foundation. We start from a Noetherian (co-well-founded) and locally confluent binary relation $\rightsquigarrow$ on A (conditions that are satisfied in all our examples). We then construct a new Noetherian binary relation $\rightsquigarrow^{\circ}$ on the cycles of $\rightsquigarrow$. An important property of this new relation is that any cycle can be written as the "merge" of a smaller cycle and one which is given by the condition of local confluence. The concrete consequence is Theorem 45, which we call *Noetherian cycle induction*: Assume we are given a property P on cycles which respects merging cycles as well as "rotating" cycles. Then, we can show P for all cycles by showing it for the empty cycle(s) and for all cycles that are given by the local confluence property.

While it is very hard to show a property directly for *all* cycles, it is much more manageable to show the property for those cycles that stem from local confluence. In any given example, when we prove local confluence, we can *choose* how these "confluence cycles" look, giving us full control over what we have to prove.

Let us get back to homotopy type theory. The combination of the two mentioned results (Theorem 13 and Theorem 45) gives us Theorem 46: Given a 1-type X and $f : A \to X$ such that $a \sim b$ implies $f(a) = f(b)$, it suffices to show that "confluence cycles" are mapped to trivial equality proofs. We apply this to show that the free higher group over a set has trivial fundamental groups. There is a family of similar statements that we also discuss and prove.

Outline of the paper. We start by clarifying the setting and recalling previous results that we build upon in Section 2. We then show the mentioned Theorem 13 describing functions out of the set-quotient into 1-types in Section 3. In Section 4, we identify the properties of the relations in

question, namely being Noetherian and local confluence. Section 5 is the core of the paper: Starting from a Noetherian locally confluent relation on A, we construct a Noetherian relation on the cycles of A. This allows us to formulate the principle of Noetherian cycle induction in Section 6. In Section 7, we show several applications. Section 9 is reserved for some concluding remarks.

Formalisation. We have formalised the complete construction with all proofs in Sections 2 to 6 in the proof assistant Lean [8], including the existing results that we recall in Section 2. Parts of Section 7 are not self-contained and have not been included in the formalisation, but the important Theorem 46 is formalised. The formalisation was done for the third major version of Lean, which supports reasoning in HoTT by enforcing that its strict, impredicative universe of propositions is avoided in all definitions. It relies on a port of the Lean HoTT library [29] which was developed for Lean 2. The Lean code is available online.[2]

The formalisation approximately follows the structure of the paper. Many arguments are directly translated from the informal mathematical style of the paper into Lean code. In some cases, the formalisation uses shortcuts compared to the paper. These shortcuts would look ad-hoc and unnecessary on paper, where certain overhead is invisible anyway. However, avoiding this overhead simplifies the formalisation.

While homotopy type theory often allows very short formalisations of "homotopical" arguments (such as Theorem 13), combinatorial arguments (such as the ones leading to Theorem 45) are more tedious in Lean than they are on paper. It is difficult to say how big the formalisation exactly is due to the use of existing results, some of which are available in libraries while others are not. Only counting the core results, the Lean formalisation comprises approximately 1.600 lines of code.

2 Preliminaries and Prerequisites

2.1 Setting

We work in the version of homotopy type theory that is developed in the "HoTT book" [27]. This means that we use a dependent type theory with Σ- and Π-types as well as coproducts, with univalent universes and function extensionality. We use inductive families [9] as for example in (11) below.[3] Further, we use curly brackets to denote implicit arguments as it is done in various proof assistants, for example as in $f : \Pi\{a : A\}.(B(a) \to C(a))$. This allows us to omit the implicit argument and write $f(b) : C(b)$ in situations where a can easily be inferred. The only higher inductive types that are relevant for this paper besides set-quotients are co-equalisers and set-truncations. They are briefly introduced

---

[2] The online repository can be found at https://gitlab.com/fplab/freealgstr.
[3] It is known (cf. [18]) that any development involving inductive families can be expressed with only ordinary inductive types.

in Section 2.3 below. (For the advanced application given by Theorem 50, pushouts [27, Chp 6.8] are used.)

2.2 Relations and Closures

As usual, we have the following closure operations:

Definition 2 (closures). Given a relation $(\sim) : A \to A \to \mathcal{U}$, we write:

- $\sim^+$ for its transitive closure,
- $\sim^*$ for its reflexive-transitive closure,
- $\sim^s$ for its symmetric closure,
- $\sim^{s*}$ for its reflexive-symmetric-transitive closure.

The type-theoretic implementation of these closures is standard. The reflexive-transitive closure is constructed as an inductive family with two constructors:

$$\text{inductive } (\sim^*) : A \to A \to \mathcal{U} \text{ where}$$

$$\text{nil} : \Pi\{a : A\}.(a \sim^* a)$$

$$\text{cons} : \Pi\{a, b, c : A\}.(a \sim^* b) \to (b \sim c) \to (a \sim^* c) \tag{11}$$

The symmetric closure is the obvious disjoint sum,

$$(a \sim^s b) :\equiv (a \sim b) + (b \sim a). \tag{12}$$

The transitive closure can be defined analogously to (11), with the constructor nil replaced by a constructor of the form $(a \sim b) \to (a \sim^+ b)$. Note that the relation $\sim$ is not required to be valued in propositions or even sets and neither are any of its closures. As a consequence, we have functions $(a \sim^* b) \to (a \sim^{**} b)$ and $(a \sim^{**} b) \to (a \sim^* b)$, but we do in general not have $(a \sim^* b) = (a \sim^{**} b)$. The analogous caveat holds for the other closure operations.

Remark 3. We often want to nest several constructions. If we start with the relation $\sim$ and first apply construction x, then y, then z, we denote the resulting relation by $\sim^{xyz}$. For example, the reflexive-symmetric-transitive closure $\sim^{s*}$ is constructed by first taking the symmetric and then the reflexive-transitive closure.

Needless to say, there are several possible variations of how such concepts can be implemented; for example, $\sim^*$ could be defined as $(x \sim^+ y) + (x = y)$. The definitions we give are the ones which we found lead to the best computational behaviour in a proof assistant.

Definition 4 (chains and cycles). Given a relation $\sim$, we refer to elements of $(a \sim^{s*} b)$ as *chains (from a to b)*. A chain is *monotone* if either each segment comes from the left summand in (12), i.e. is of the form $\text{inl}(t)$ with $t : a \sim b$ (no segment is inverted), or if segment comes from the right summand (every segment is inverted).

An element of $(a \sim^{s*} a)$ is called a *cycle (based at a)*. We write $\bigcirc^{\sim}$ for the type of cycles,

$$\bigcirc^{\sim} :\equiv \Sigma(a : A).(a \sim^{s*} a). \tag{13}$$

Fig. 1 below illustrates these definitions, where $a \to b$ denotes $a \sim b$.

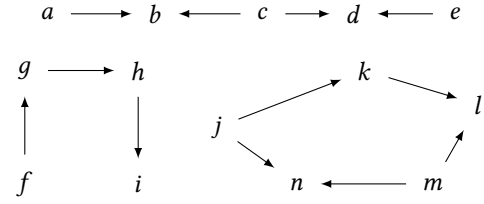**Figure 1.** A chain from a to e, a monotone chain from f to i, and a cycle. We sometimes call the elements of A *vertices*.

Definition 5 (notation and operations on closures). We have the following standard operations:

1. *reflexivity:* We write $\epsilon : a \sim^* a$ for the constructor nil. We also write ϵ for the trivial cycle, and ϵ_a if we want to emphasise that we mean the trivial cycle based at a.
2. *transitivity:* Given $\alpha : a \sim^* b$ and $\beta : b \sim^* c$, we notate their concatenation by $(\alpha :: \beta) : a \sim^* c$. Overloading the symbol $::$, we also use it instead of the constructor cons.
3. *symmetry:* We can invert $\alpha : a \sim^{s*} b$ to get $\alpha^{-1} : b \sim^{s*} a$. This is done by inverting each single segment and reversing the order of all segments.

Definition 6 (empty chains and cycles). There is an obvious function

$$\text{length} : \Pi\{a, b : A\}.(a \sim^{s*} b) \to \mathbb{N} \tag{14}$$

which calculates the length of a chain. We say that a chain α is *empty* if its length is zero. For the type of *empty cycles*, we write

$$\bigcirc_{\emptyset}^{\sim} :\equiv \Sigma(\alpha : \bigcirc^{\sim}).\text{length}(\alpha) = 0. \tag{15}$$

Remark 7. The trivial cycle ϵ is always empty, but not every empty cycle is equal to ϵ. Instead, an empty cycle corresponds to a loop in A,

$$(\Sigma(a : A).(a = a)) \simeq \bigcirc_{\emptyset}^{\sim}. \tag{16}$$

This is easy to see, since (11) without the second constructor is the usual definition of Martin-Löf's identity type as an inductive family. In particular, if A is a set, then ϵ *is* the only empty cycle.

2.3 Quotients, Coequalisers, Truncations

As explained in the introduction, the *set-quotient* $A/\sim$ is the higher inductive type with constructors ι, glue, and trunc, see (1). The construction can be split into two steps. We write $(A /\!\!/ \sim)$ for the *untruncated quotient* or *coequaliser* which has only the constructors ι and glue, and we write $\|-\|_0$ for the *set-truncation* which has only the constructors ι and trunc.

Lemma 8. *For a relation $\sim$ on A, we have*

$$(A/\sim) \simeq \left\| A /\!\!/ \sim \right\|_0. \tag{17}$$

Proof. The direct approach of constructing functions back and forth works without difficulties.   □

For a given type X, there is a canonical function from the function type $(A /\!/_\sim) \to X$ to the Σ-type of pairs (f, h), where

$$f : A \to X, \qquad (18)$$

$$h : \Pi\{a, b : A\}.(a \sim b) \to f(a) = f(b). \qquad (19)$$

This map is given by:

$$g \mapsto (g \circ [-], \mathsf{ap}_g \circ \mathsf{glue}). \qquad (20)$$

The universal property of the higher inductive type $A /\!/_\sim$ tells us that this function is an equivalence (one can of course also show this with the dependent elimination principle of $A /\!/_\sim$, if that is assumed instead as primitive).

2.4 Path Spaces of Coequalisers

We will need to prove statements about equalities in coequalisers. For this, we use the following result which characterises the path spaces of $(A /\!/_\sim)$:

Theorem 9 (induction for coequaliser equality, [21]). *Let a relation* $(\sim) : A \to A \to \mathcal{U}$ *as before and a point* $a_0 : A$ *be given. Assume we further have a type family*

$$P : \Pi\{b : A\}.(\iota(a_0) =_{A/\!/_\sim} \iota(b)) \to \mathcal{U} \qquad (21)$$

together with terms

$$r : P(\mathsf{refl}_{\iota(a_0)}), \qquad (22)$$

$$e : \Pi\{b, c : A\}, (q : \iota(a_0) = \iota(b)), (s : b \sim c).$$

$$P(q) \simeq P(q \cdot \mathsf{glue}(s)). \qquad (23)$$

Then, we can construct a term

$$\mathsf{ind}_{r,e} : \Pi\{b : A\}, (q : \iota(a_0) = \iota(b)).P(q) \qquad (24)$$

with the following β-*rules:*

$$\mathsf{ind}_{r,e}(\mathsf{refl}_{\iota(a_0)}) = r, \qquad (25)$$

$$\mathsf{ind}_{r,e}(q \cdot \mathsf{glue}(s)) = e(q, s, \mathsf{ind}_{r,e}(q)). \qquad (26)$$

$\square$

2.5 Functions out of Set-Truncations

For types A and B, we have a canonical function

$$(\|A\|_0 \to B) \to (A \to B) \qquad (27)$$

which is given by precomposition with $|-|_0$. Any such function $g \circ |-|_0$ is moreover constant on loop spaces in the sense that

$$\mathsf{ap}_{g \circ |-|_0} : (a = a) \to (g(a) = g(a)) \qquad (28)$$

satisfies $\mathsf{ap}_{g \circ |-|_0}(p) = \mathsf{refl}$, for all a and p. For a 1-truncated type B, the following known result by Capriotti, Kraus, and Vezzosi states that this property is all one needs to reverse (27):

Theorem 10 ([5]). *Let A be a type and B be a 1-truncated type. The canonical function from* $(\|A\|_0 \to B)$ *to the type*

$$\Sigma(f : A \to B).\Pi(a : A), (p : a = a).\mathsf{ap}_f(p) = \mathsf{refl} \qquad (29)$$

is an equivalence. $\square$

3 Eliminating out of Set-Quotients

As before, let $\sim$ be a relation on A. Assume further that given are a function $f : A \to X$ and a proof h that f sends related points to equal points, as in (18) and (19). There is an obvious function

$$h^{s*} : \Pi\{a, b : A\}.(a \sim^{s*} b) \to f(a) = f(b), \qquad (30)$$

defined by recursion on $a \sim^{s*} b$ which in each step path-composes with an equality given by h or the inverse of such an equality. Given (f, h) and a third map $k : X \to Y$, it is easy to prove by induction on $a \sim^* b$ that we have

$$\mathsf{ap}_k \circ h^{s*} = (\mathsf{ap}_k \circ h)^{s*}. \qquad (31)$$

We also note that, for chains α, β,

$$h^{s*}(\alpha :: \beta) = h^{s*}(\alpha) \cdot h^{s*}(\beta) \text{ and} \qquad (32)$$

$$h^{s*}(\alpha^{-1}) = (h^{s*}(\alpha))^{-1}. \qquad (33)$$

Of particular interest is the function $\mathsf{glue}^{s*} : \Pi\{a, b : A\}.(a \sim^{s*} b) \to \iota(a) = \iota(b)$. It is in general not an equivalence: For example, for $t : a \sim a$, the chain $t :: t^{-1}$ and the empty chain both get mapped to refl. Thus, glue^{s*} does not preserve inequality (but see Remark 7). However, we have the following result:

Lemma 11. *The function* $\mathsf{glue}^{s*} : (a \sim^{s*} b) \to \iota(a) = \iota(b)$ *is surjective.*

Proof. Fixing one endpoint $a_0 : A$ and setting

$$P : \Pi\{b : A\}.(\iota(a_0) = \iota(b)) \to \mathcal{U} \qquad (34)$$

$$P(q) :\equiv \|\Sigma(p : a_0 \sim^{s*} b).\mathsf{glue}^{s*}(p) = q\|_{-1} \qquad (35)$$

we need to show that, for all q, we have $P(q)$. We use Theorem 9, where r is given by the empty chain. To construct e, we need to prove $P(q) \simeq P(q \cdot \mathsf{glue}(s))$ for any $s : b \sim c$. This amounts to constructing functions in both directions between the types $\Sigma(p : a_0 \sim^{s*} b).\mathsf{glue}^{s*}(c) = q$ and $\Sigma(p : a_0 \sim^{s*} b).\mathsf{glue}^{s*}(c) = q \cdot \mathsf{glue}(s)$, where extending a chain with s or with s^{-1} is sufficient. $\square$

The following is a "derived induction principle" for equalities in coequalisers:

Lemma 12. *For a family* $P : \Pi\{x : A /\!/_\sim\}.x = x \to \mathcal{U}$ *such that each $P(q)$ is a proposition, the two types*

$$\Pi(\gamma : \bigcirc^\sim).P(\mathsf{glue}^{s*}(\gamma)). \qquad (36)$$

and

$$\Pi(x : A /\!/_\sim), (q : x = x).P(q) \qquad (37)$$

are equivalent.

Proof. Both types are propositions, and the second clearly implies the first. For the other direction, induction on x lets us assume that x is of the form $\iota(a)$ for some $a : A$; the case for the constructor glue is automatic. The statement then follows from the surjectivity of glue^{s*}. $\square$

Theorem 13. *Let $A : \mathcal{U}$ be a type, $(\sim) : A \to A \to \mathcal{U}$ be a relation, and $X : \mathcal{U}$ be a 1-type. Then, the type of functions $(A/\sim \to X)$ is equivalent to the type of triples (f, h, c) (a nested Σ-type), where*

$$f : A \to X \tag{38}$$

$$h : \Pi\{a, b : A\}.(a \sim b) \to f(a) = f(b) \tag{39}$$

$$c : \Pi(\gamma : \bigcirc^\sim).h^{s*}(\gamma) = \mathsf{refl}. \tag{40}$$

Proof. We have the following chain of equivalences:

$$A/\sim \to X$$

$\simeq$ *by Lemma* 8

$$\left\| A /\!\!/ \sim \right\|_0 \to X$$

$\simeq$ *by Theorem* 10

$$\Sigma g : (A /\!\!/ \sim) \to X.$$
$$c : \Pi\{x : A /\!\!/ \sim\}, (q : x = x).\mathsf{ap}_g(q) = \mathsf{refl}$$

$\simeq$ *by Lemma* 12

$$\Sigma g : (A /\!\!/ \sim) \to X.$$
$$c : \Pi(\gamma \bigcirc^\sim).\mathsf{ap}_g(\mathsf{glue}^{s*}(\gamma)) = \mathsf{refl}$$

$\simeq$ *by* (31)

$$\Sigma g : (A /\!\!/ \sim) \to X.$$
$$c : \Pi(\gamma : \bigcirc^\sim).(\mathsf{ap}_g \circ \mathsf{glue})^{s*}(\gamma) = \mathsf{refl}$$

$\simeq$ *by* (20)

$$\Sigma f : A \to X.$$
$$\Sigma h : \Pi\{a, b : A\}.(a \sim b) \to f(a) = f(b).$$
$$c : \Pi(\gamma : \bigcirc^\sim).h^{s*}(\gamma) = \mathsf{refl}$$

$\square$

4   On Confluence and Well-Foundedness

In the theory of *rewriting systems*, a usually desirable property of a rewriting relation is *strong normalisation*, meaning that any term can be rewritten to exactly one irreducible term. The relations that we are interested in are much weaker. For a rewriting system, it is usually decidable whether and to what a term can be rewritten. In contrast, it is in the examples discussed in the introduction generally undecidable whether, for a given $a : A$, there is a b such that $a \sim b$. Even if we already have both a and b, it is undecidable whether $a \sim b$. Nevertheless, the concepts of *confluence* and *well-foundedness* make sense and are highly useful in our setting to solve cases such as the ones from the introduction.

Convention 14. To emphasise the "directedness" of a relation, we name relations $\rightsquigarrow$ instead of $\sim$ when talking about confluence. This is only to be understood as supporting the intuition, it does not come with any implicit assumptions. Similarly, we use the relation name $<$ in the context of well-foundedness. We also use the name *order* synonymously with *relation*; note that an order is still simply a type family $A \to A \to \mathcal{U}$, again without any implicit assumptions.

Definition 15 (span). Given a relation $\rightsquigarrow$, a *span* is a 5-tuple (a, b, c, s, t) of $a, b, c : A$ together with $s : a \rightsquigarrow b$ and $t : a \rightsquigarrow c$. We write $(\cdot \leftsquigarrow \cdot \rightsquigarrow \cdot)$ for the type of spans. If b and c are fixed, we write $(b \leftsquigarrow \cdot \rightsquigarrow c)$ for the type of triples (a, s, t). And *extended span* is the same 5-tuple as s span, but with $s : a \rightsquigarrow^* b$ and $t : a \rightsquigarrow^* c$. The notion of a *cospan* and an *extended cospan* are defined analogously, with the directions of s and t reversed. We use the notations $(\cdot \;{}^*\!\!\leftsquigarrow \cdot \rightsquigarrow^* \cdot)$, and $(b \rightsquigarrow^* \cdot \;{}^*\!\!\leftsquigarrow c)$ and so on in the obvious way.

Definition 16 (confluence). We say that a relation $\rightsquigarrow$ on A is *locally confluent* if, for any span, there is a matching extended cospan:

$$\mathsf{locConf}(\rightsquigarrow) :\equiv \Pi\{b, c : A\}.(b \leftsquigarrow \cdot \rightsquigarrow c) \to (b \rightsquigarrow^* \cdot \;{}^*\!\!\leftsquigarrow c). \tag{41}$$

We say that $\rightsquigarrow$ is *confluent* if we can replace the assumption $(b \leftsquigarrow \cdot \rightsquigarrow c)$ by the weaker assumption $(b \;{}^*\!\!\leftsquigarrow \cdot \rightsquigarrow^* c)$.

Remark 17. Note that *being [locally] confluent* is not a proposition; it should be understood as *carrying a [local] confluence structure*.

The definition of *well-foundedness* is standard as well. If $x < y$, we say that x is smaller than y. Recall that a point $a : A$ is $<$-*accessible* if every $x < a$ is $<$-accessible, and $<$ is *well-founded* if all points are $<$-accessible. Type-theoretically, this can be expressed as follows:

Definition 18 ($\Phi_<$ in [2]). The family $\mathsf{acc}^< : A \to \mathcal{U}$ is generated inductively by a single constructor,

$$\mathsf{step} : \Pi(a : A).(\Pi(x : A).(x < a) \to \mathsf{acc}^<(x)) \to \mathsf{acc}^<(a) \tag{42}$$

Further, we define

$$\mathsf{isWellFounded}(<) :\equiv \Pi(a : A).\mathsf{acc}^<(a). \tag{43}$$

While the definition in [27, Chp. 10.3] is only given for the special case that A is a set and $<$ is valued in propositions, the more general case that we consider works in exactly the same way (cf. our formalisation). In particular, we have the following two results:

Lemma 19. *For any x, the type $\mathsf{acc}^<(x)$ is a proposition. Further, the statement that $<$ is well-founded is a proposition.* $\square$

Lemma 20 (accessibility induction [27, Chp. 10.3]). *Assume we are given a family $P : A \to \mathcal{U}$ such that we have*

$$\Pi(a_0 : A).\mathsf{acc}^<(a_0) \to (\Pi(a < a_0).P(a)) \to P(a_0). \tag{44}$$

In this case, we get:

$$\Pi(a_0 : A).\mathrm{acc}^<(a_0) \to P(a_0). \qquad (45)$$

If $<$ is well-founded, the argument $\mathrm{acc}^<(a_0)$ can be omitted and the principle is known as well-founded induction.

An immediate application is the following:

Lemma 21. If $a < a$, then a is not $<$-accessible.

Proof. $<$-accessibility induction with the family $P(a) :\equiv (a < a) \to \emptyset$. □

Lemma 22. If $a : A$ is $<$-accessible, then it is $<^+$-accessible.

Proof. By $<$-accessibility induction on $P(a) :\equiv (\mathrm{acc}^<a) \to (\mathrm{acc}^{<^+}a)$. □

The notion relevant to our examples and application is not well-foundedness, but co-well-foundedness: instead of "no infinite sequences to the left" the property we want to use is "no infinite sequences to the right". Such a relation is known as *Noetherian* in the literature. Of course, this amounts to a simple swap of arguments.

Definition 23 (Noetherian). A relation $>$ is called *Noetherian* (or *co-well-founded*) if the opposite relation $>^{\mathrm{op}}$, defined by $(b >^{\mathrm{op}} a) :\equiv (a > b)$, is well-founded. The corresponding induction principle is known as *Noetherian induction*.

We have already seen an example in the introduction of this paper. Recall the relation (3) that is used in the construction of the free group in Example 1: Given two lists $\ell_1, \ell_2 : \mathrm{List}(A + A)$, we have $\ell_1 \rightsquigarrow \ell_2$ if the first list can be transformed into the second list by removing exactly two elements. The two removed list elements have to be consecutive and "inverse" to each other, i.e. one is of the form $\mathrm{inl}(a)$, the other $\mathrm{inr}(a)$.

Lemma 24 (free groups, continuing Example 1). *The relation $\rightsquigarrow$ on lists, in (3) named $\sim$, is Noetherian and locally confluent.*

Proof. Noetherian is trivial since each step decreases the length of the list. Regarding local confluence, assume that we have a list ℓ which contains two redexes (x, x^{-1}) and (y, y^{-1}). We write ℓ_x for the list with the first redex removed and ℓ_y for the list with the second redex removed. There are only three cases:

1. The two redexes are the same (they "fully overlap"). In this case, there is nothing to do: The extended cospan is empty.
2. The two redexes partially overlap, in the sense that $x^{-1} = y$ (or $y^{-1} = x$, which is equivalent). In this case, we again have $\ell_x = \ell_y$.
3. There is no overlap between the two redexes. We can then remove the redex (y, y^{-1}) from ℓ_x and have constructed a list equal to the one we get if we remove (x, x^{-1}) from ℓ_y.

□

Corollary 25 (of Lemma 22). *If $<$ is well-founded [Noetherian], then so is $<^+$.*

It is a classic result by by Newman [24] that local confluence is as strong as general confluence in the context of Noetherian relations. The proof by Huet [15] can directly be expressed in our setting. Strictly speaking, our formulation could be described as stronger since we do not assume that the relation is decidable or even given as a function, but Huet's argument stays exactly the same:

Theorem 26 (Newman [24], Huet [15]). *A locally confluent and Noetherian relation $\rightsquigarrow$ is confluent.*

Proof. By Noetherian induction, with $P(u)$ expressing that any extended span $s \;{}^*\!\!\leftsquigarrow \cdot \rightsquigarrow^* t$ can be completed to a quadrangle. If either leg has length 0, the statement is trivial. Otherwise, we can write it as $s^*\!\!\leftsquigarrow v \leftsquigarrow u \rightsquigarrow w \rightsquigarrow^* t$. By local confluence, we get x such that $v \rightsquigarrow^* x$ and $w \rightsquigarrow^* x$. By $P(v)$, we get y with $s \rightsquigarrow^* y$ and $x \rightsquigarrow y$. By $P(w)$, we get z with $y \rightsquigarrow^* z$ and $t \rightsquigarrow^* z$. This proof can be pictured as shown below.

□
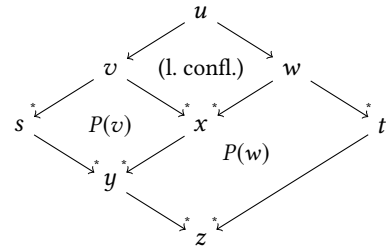
5 Ordering Cycles

Let again $<$ be a relation on A. The aim of this section is to construct a relation on cycles $\bigcirc^<$. We proceed in several steps. We start by defining a relation on ordinary lists, i.e. on the type $\mathrm{List}\,A$, similar to the well-known *multiset extension*. Next, we extend the relation to *rotating* lists. Finally, we construct the relation on cycles and prove essential properties.

5.1 An Order on Lists

The type of *lists on A* is, as it is standard, inductively generated by a constructor nil for the empty list and a constructor adding a single element, and it is written as $\mathrm{List}\,A$. As for chains, we write $::$ for both adding a single element and for list concatenation. We write $[a]$ for the list of length 1 with the single element $a : A$.

Definition 27. Given the relation $<$ as before, we define a relation $<^L$ on $\mathrm{List}\,A$: For lists k, l, we define $k <^L l$ to mean that k can be constructed from l by replacing a single list element x in l by a finite number of elements of A, all of which are smaller than x.

A type-theoretic implementation of this relation is the following. For $k : \text{List } A$ and $x : A$, we first define $k <^{\text{all}} x$ as "every element y of k satisfies $y < x$". The relation $<^L$ is then generated inductively by a single constructor as in

$$\text{inductive } (<^L) : \text{List } A \to \text{List } A \to \mathcal{U}$$
$$\text{step} : \Pi\{k, l_1, l_2 : \text{List } A\}.\Pi\{x : A\}. \tag{46}$$
$$(k <^{\text{all}} x) \to (l_1 :: k :: l_3) <^L (l_1 :: x :: l_3)$$

The following strengthening of the induction principle is derivable:

Lemma 28 (nested induction). *Assume we are given a relation $<_1$ on a type B, a relation $<_2$ on a type C, and a family $P : B \times C \to \mathcal{U}$. Assume further that we are given*

$$\Pi(b : B, c : C).\text{acc}^{<_1}(b) \to \text{acc}^{<_2}(c) \to$$
$$(\Pi(b' <_1 b).P(b', c)) \to (\Pi(c' <_2 c).P(b, c')) \to \tag{47}$$
$$P(b, c).$$

Then, we get:

$$\Pi(b : B, c : C).\text{acc}^{<_1}(b) \to \text{acc}^{<_2}(c) \to P(b, c). \tag{48}$$

Proof. By double induction on the accessibility witnesses. □

The next three short lemmas show that $<^L$ is well-founded assuming that $<$ is. The argument mirrors Nipkov's proof that the multiset extension preserves well-foundedness [25].

Lemma 29. *If lists l and k are both $<^L$-accessible, then so is $l :: k$.*

Proof. We use Lemma 28 with List A and $<^L$ as both $<_1$ and $<_2$, and with $P(l, k) :\equiv \text{acc}^{<^L}(l :: k)$. We need to show (47). Assume l and k are $<^L$-accessible. By definition, $P(l, k)$ holds if every list smaller than $l :: k$ is accessible. By Definition 27, a smaller list can either be written as $l' :: k$ with $l' <^L l$ or as $l :: k'$ with $k' <^L k$. These are accessible by induction hypothesis. □

Lemma 30. *If a is $<$-accessible, then the singleton list $[a]$ is $<^L$-accessible.*

Proof. We do $<$-induction on a. It is enough to show that $l \equiv [a_1, a_2, \ldots, a_n]$ is $<^L$-accessible for any $l <^L [a]$. By the induction hypothesis, every $[a_i]$ is $<^L$-accessible. By Lemma 29, the whole list is $<^L$-accessible. □

Lemma 31. *If $<$ is well-founded, then so is $<^L$.*

Proof. Writing a given list l as the concatenation of singleton lists and applying Lemma 30, the list l is $<^L$-accessible. □

5.2 A Rotation-Invariant Relation on Lists

Rotating a list is defined in the obvious way: The function

$$\text{rot} : \text{List } A \to \text{List } A \tag{49}$$

removes the first element of a list (if it exists) and adds it at the end of a list. We now define a version of $<^L$ which does not discriminate between lists that are the same module rotation.

Definition 32. Given the relation $<$ on A, we define the relation $<^\bigcirc$ on List A by setting

$$(k <^\bigcirc l) :\equiv \Sigma(n : \mathbb{N}).\text{rot}^n(k) <^L l. \tag{50}$$

By definition, the relation $<^\bigcirc$ is invariant under arbitrarily rotating the smaller list. We also have this property for the larger list:

Lemma 33. *If we have $k <^\bigcirc l$, then we also have that $k <^\bigcirc \text{rot}^m(l)$ for any natural number m.*

Proof. It is enough to show the statement for $m \equiv 1$. Let us assume that we have a number n such that $\text{rot}^n(k) <^L l$. By definition of $<^L$, the latter expression can be written as

$$(l_1 :: l_2 :: l_3) <^L (l_1 :: x :: l_3). \tag{51}$$

If we rotate the right-hand side once, then there are two cases: If l_1 is non-empty, then rotating the left-hand side once gives again an instance of $<^L$. If l_1 is the empty list, then rotating the left-hand side length(k) times yields an instance of $<^L$. □

Corollary 34. *A list l is $<^\bigcirc$-accessible if and only if $\text{rot}(l)$ is.* □

Lemma 35. *If a list l is $<^L$-accessible, then it is $<^\bigcirc$-accessible.*

Proof. We do $<^L$-accessibility induction on l and assume $\text{acc}^{<^L}(l)$. This allows us to assume that any k with $k <^L l$ is $<^\bigcirc$-accessible.

Given any k' with $k' <^\bigcirc l$, we need to show that k' is $<^\bigcirc$-accessible. By definition, we have a number n such that $\text{rot}^n(k') <^L l$. Therefore, $\text{rot}^n(k')$ and by Corollary 34 is $<^\bigcirc$-accessible. □

Corollary 36. *If the relation $<^L$ is well-founded, then so is $<^\bigcirc$.*

5.3 An Order on Cycles

Rotating a chain is defined analogously to rotating a list. Specialising to cycles and re-using the name introduced at (49), this gives us a function

$$\text{rot} : \bigcirc^< \to \bigcirc^<. \tag{52}$$

This function essentially rotates the base point of the cycle.

For a given chain, we can produce the list of its vertices and forget the actual relation. We define this function as follows:

$$\pi : (a \rightsquigarrow^{s*} b) \to \text{List } A$$
$$\pi(\text{nil}_a) :\equiv \text{nil} \tag{53}$$
$$\pi(\text{cons}_{a,b,c}(f, p)) :\equiv \pi(f) :: c$$

Specialised to cycles, this function

$$\pi : \bigcirc^< \to \text{List } A \tag{54}$$

lists the vertices of a cycle. Clearly, the functions rot and π commute. Note that the function π ignores the very first endpoint of a chain: $\pi(\mathrm{nil}_a)$ is the empty list, *not* the list containing a. This ensures that π works on cycles as expected and does not list the base point twice. (It also means that an empty cycle, although still being based at some point, has an empty list of vertices. This awkward edge case will not be relevant.)

We can now extend the relation $<^L$ to chains and $<^\circ$ to cycles, overloading the symbols:

Definition 37 (inherited relations on chains and cycles). For chains $f : a <^{s*} b$ and $g : c <^{s*} d$, we define $f <^L g$ to mean $\pi(f) <^L \pi(g)$. For cycles $p, q : \bigcirc^<$, we define $p <^\circ q$ to mean $\pi(p) <^\circ \pi(q)$.

Theorem 38. *If $\rightsquigarrow$ is Noetherian on A, then $\rightsquigarrow^{+\circ+}$ is Noetherian on $\bigcirc^{\rightsquigarrow}$.*

Proof. By combining Lemma 31 with Corollaries 25 and 36. □

5.4 Constructions with Noetherian Relations on Cycles

Definition 39 (span). Let $\gamma : \bigcirc^{\rightsquigarrow}$ be a cycle. We say that γ *contains a span* if we find consecutive vertices $v \leftsquigarrow u \rightsquigarrow w$, i.e. if we have

$$\gamma = (\alpha :: s^{-1} :: t :: \beta) \tag{55}$$

for some $\alpha : a \rightsquigarrow^{s*} v$, $s : u \rightsquigarrow v$, $t : u \rightsquigarrow w$, $\beta : w \rightsquigarrow^{s*} a$.

The very intuitive statement of the next lemma is illustrated in Fig. 2 below.

Lemma 40. *If a relation $\rightsquigarrow$ is well-founded (or Noetherian), then any cycle $\gamma : \bigcirc^{\rightsquigarrow}$ is either empty or contains a span.*

Proof. If α is empty, we are done. Otherwise, by Lemma 22 combined with Lemma 21, γ is not monotone in the sense of Definition 4, i.e. not all segments of γ go into the same direction. This means that we can find a span by going over the vertices of the cycle. □
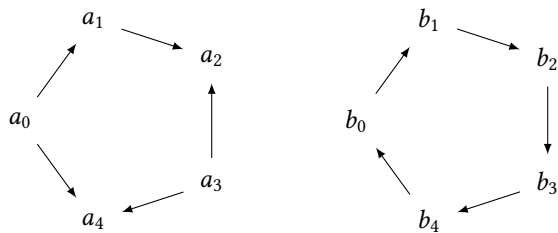

Figure 2. Two elements of $\bigcirc^<$. The left cycle consists of points $a_i : A$ and contains two spans, namely $a_4 \leftsquigarrow a_0 \rightsquigarrow a_1$ and $a_2 \leftsquigarrow a_3 \rightsquigarrow a_4$. The right cycle does not contain any span; this is not possible if the relation is Noetherian.

Definition 41 (merge of cycles). Assume we have $a, b : A$ and $\alpha, \beta, \gamma : a \sim^* b$. We say that the cycle $\alpha :: \gamma^{-1}$ is the *merge* of the cycles $\alpha :: \beta^{-1}$ and $\beta :: \gamma^{-1}$.
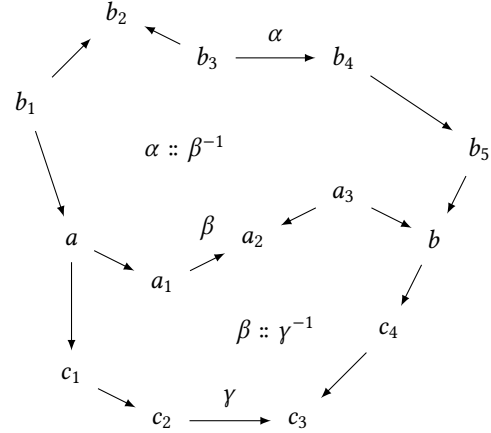

Figure 3. The *merge* of cycles as in Definition 41. The big cycle is the merge of the two smaller ones. Technically, any of the three cycles is the merge of the other two.

Let $\rightsquigarrow$ be locally confluent. By Definition 16, we get a map from spans to extended cospans. By concatenating the original span with this extended cospan, we get a cycle. We denote this function from spans to cycles by

$$\mathfrak{L} : (\cdot \leftsquigarrow \cdot \rightsquigarrow \cdot) \rightarrow \bigcirc^{\rightsquigarrow} \tag{56}$$

and call a cycle of the form $\mathfrak{L}(\kappa)$ a *confluence cycle*.

Lemma 42. *Let $\rightsquigarrow$ be Noetherian and locally confluent. Any cycle γ is either empty or there is a number n such that the rotation $\mathrm{rot}^n(\gamma)$ can be written as the merge of a confluence cycle and a cycle δ with $\gamma \rightsquigarrow^{+\circ+} \delta$.*

Proof. If γ is non-empty then, by Lemma 40, it contains a span. We can then write γ as given by (55). By rotating, we can bring the cycle into the form $\mathrm{rot}^n(\gamma) = (\kappa :: \tau^{-1})$ with $\kappa : v \leftsquigarrow u \rightsquigarrow w$ being the span and $\tau : v \rightsquigarrow^{s*} w$ being a chain. By local confluence, there is a chain $\alpha : v \rightsquigarrow^{s*} w$ which is smaller than the span in the sense that $\kappa \rightsquigarrow^{+L+} \alpha$. We define $\delta :\equiv \alpha \circ \tau^{-1}$ and have $\mathrm{rot}^n(\gamma) \rightsquigarrow^{+\circ+} \delta$, thus by Lemma 33 $\alpha \rightsquigarrow^{+\circ+} \delta$ as required. □

Remark 43. With the construction in the above proof, we have in almost all cases $\gamma \rightsquigarrow^{+\circ} \delta$. Only if the chain α is empty, which may happen if $v = w$, we need two steps from γ to δ.

6 Families over Cycles

We move on from studying cycles to exploring type families indexed over cycles.

Definition 44 (stability under merging and rotating). Assume that $Q : \bigcirc^{\rightsquigarrow} \to \mathcal{U}$ is a family of types. We say that Q is *stable under merging* if, whenever γ is the merge of cycles α and β such that we have $Q(\alpha)$ and $Q(\beta)$, we also have $Q(\gamma)$. We say that Q is *stable under rotating* if, for any cycle δ such that $Q(\delta)$, we also have $Q(\text{rot}(\delta))$.

Theorem 45 (Noetherian cycle induction). *Let $\rightsquigarrow$ be Noetherian and locally confluent. Assume further that $Q : \bigcirc^{\rightsquigarrow} \to \mathcal{U}$ is stable under merging and rotating. If Q is inhabited at every empty cycle and at every confluence cycle, then Q is inhabited everywhere:*

$$\Pi(\gamma : \bigcirc^{\rightsquigarrow}_{\emptyset}).Q(\gamma)$$
$$\to \quad \Pi(\kappa : \cdot \curvearrowleft \cdot \rightsquigarrow \cdot).Q(\mathfrak{L}(\kappa)) \qquad (57)$$
$$\to \quad \Pi(\alpha : \bigcirc^{\rightsquigarrow}).Q(\alpha).$$

If A is a set, then the first line of the above principle can be replaced and it can be stated as follows:

$$\Pi(a : A).Q(\epsilon_a)$$
$$\to \quad \Pi(\kappa : \cdot \curvearrowleft \cdot \rightsquigarrow \cdot).Q(\mathfrak{L}(\kappa)) \qquad (58)$$
$$\to \quad \Pi(\alpha : \bigcirc^{\rightsquigarrow}).Q(\alpha).$$

Proof. The relation $\rightsquigarrow^{+\bigcirc+}$ is Noetherian by Theorem 38. We perform Noetherian induction. The induction hypothesis in particular tells us that Q is inhabited for the smaller cycle given by Lemma 42. From this and from the assumption that Q is inhabited for every confluence cycle, we get the desired result. The version for A being a set follows from Remark 7. □
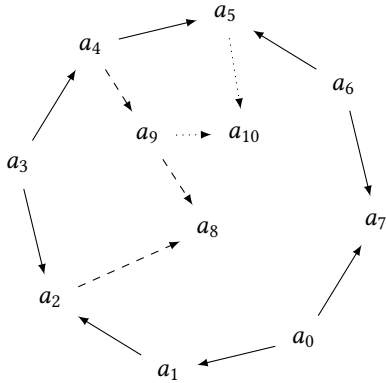

Figure 4. An illustration of Noetherian cycle induction. Assume we want to show a property Q for the big octagon $a_0 - a_7$. We first "remove" the confluence cycle spanned by $a_2 \curvearrowleft a_3 \rightsquigarrow a_4$ to get the nonagon consisting of $a_0 - a_9$ without a_3 but with the dashed edges: we now have more vertices, but the nonagon is smaller than the octagon in the order $\rightsquigarrow^{+\bigcirc+}$. In the next step, we "remove" the confluence cycle spanned by $a_9 \curvearrowleft a_4 \rightsquigarrow a_5$, and so on. Which confluence cycle is removed in a certain step depends on the precise proof of Lemma 40.

7 Applications of Noetherian Cycle Induction

We are now ready to combine the theory developed in the previous two sections with our result on set-quotients.

Theorem 46. *Let $A : \mathcal{U}$ be a type, $\rightsquigarrow: A \to A \to \mathcal{U}$ be a Noetherian and locally confluent relation, and $X : \mathcal{U}$ be a 1-type. Then, the type of functions $(A/\rightsquigarrow) \to X$ is equivalent to the type of tuples (f, h, d_1, d_2) (a nested Σ-type), where*

$$f : A \to X \qquad (59)$$
$$h : \Pi\{a, b : A\}.(a \rightsquigarrow b) \to f(a) = f(b) \qquad (60)$$
$$d_1 : \Pi\{a : A\}.\Pi(p : a = a).\text{ap}_f(p) = \text{refl} \qquad (61)$$
$$d_2 : \Pi(\kappa : \cdot \curvearrowleft \cdot \rightsquigarrow \cdot).h^{s*}(\mathfrak{L}(\kappa)) = \text{refl}. \qquad (62)$$

Further, if A is a set, the type $(A/\rightsquigarrow) \to X$ is equivalent to the type of triples (f, h, d_2).

Proof. The case for A being a set follows immediately from the main statement, since the type of d_1 becomes contractible.

For the main statement, we want to apply Theorem 13. We need to show that the type of c in (40) is equivalent to the type of pairs (d_1, d_2) above. Note that they are all propositions. From c, we immediately derive (d_1, d_2) from Remark 7.

Let us assume we are given (d_1, d_2). We need to derive c. We want to apply Noetherian cycle induction as given by Theorem 45 with $Q(\gamma) :\equiv h^{s*}(\gamma) = \text{refl}$. First of all, we need to check that Q is stable under merging and rotating. Rotating is easy and follows from the fact that left- and right-inverses of path composition coincide. Merging follows from associativity of path composition and (32, 33). We refer to our Lean formalisation for the details.

Finally, we check the two assumption of Theorem 45. Let γ be an empty cycle. With Remark 7, we can construct $Q(\gamma)$ from d_1. Let κ be a span. What we need is given by d_2. □

We want to use this theorem to show that the free higher group F_A has trivial fundamental groups. Recall that this is the example discussed in the introduction, with F_A defined in equation (5).

Theorem 47. *The fundamental groups of the free higher group on a set are trivial. In other words, for a set A and any $x : F_A$, we have*

$$\pi(F_A, x) = \mathbf{1}. \qquad (63)$$

We split the proof into several small lemmas. We keep using the relation $\rightsquigarrow$ of Example 1 and Lemma 24. Further, recall the functions ω_1 (7) and ω_2 (8) from the introduction, as well as the map ω (9).

Lemma 48 (free group; continuing Example 1 and Lemma 24). *For the local confluence constructed in Lemma 24, we can construct a term*

$$d_2 : \Pi(\kappa : \cdot \curvearrowleft \cdot \rightsquigarrow \cdot).\omega_2^{s*}(\mathfrak{L}(\kappa)) = \text{refl}. \qquad (64)$$

Proof. This lemma allows a proof of the (for HoTT typical) form "generalise and do path induction". Unfortunately, it is somewhat tedious. The function $\omega_1 : \mathsf{List}(A + A) \to F_A$, cf. (7), factors as

$$\mathsf{List}(A + A) \longrightarrow \mathsf{List}(\mathsf{base} = \mathsf{base}) \longrightarrow \mathsf{base} = \mathsf{base} \quad (65)$$

where the first map applies loop on every list element, while the second concatenates; note that $F_A :\equiv (\mathsf{base} = \mathsf{base})$. The function ω_2 (8) can then be factored similarly.

We treat the three cases in Lemma 24 separately. The third case is the hardest: We need to show that, for a given list of the form $[\ldots, x, x^{-1}, \ldots, y, y^{-1}, \ldots]$, the two ways of simplifying the list get mapped to equal 2-paths between parallel 1-paths. Using (65), we can instead assume that we are given a list of loops around base. We first repeatedly use that associativity of path composition is coherent (we have "MacLane's pentagon" by trivial path induction). Then, we have to show that the two canonical ways of simplifying $e_1 \cdot (p \cdot p^{-1}) \cdot e_2 \cdot (q \cdot q^{-1}) \cdot e_3$ to $e_1 \cdot e_2 \cdot e_3$ are equal. The usual strategy in HoTT is now to generalise to the case that p and q are equalities with arbitrary endpoints rather than loops, and then do path induction. If p and q are both refl, then both simplifications become refl as well. Instead of path induction, it seems possible to prove this lemma only using naturality and Eckmann-Hilton [27, Thm 2.6.1].

In the second case of Lemma 24, we have to show that the two ways of reducing $e_1 \cdot p \cdot p^{-1} \cdot p \cdot e_2$ to $e_1 \cdot p \cdot e_2$ are equal. Again, if p is refl, this is automatic. The first case is trivial. $\qquad\square$

Lemma 49. *The free higher group F_A is a retract of $\mathsf{List}(A + A) /\!\!/ \rightsquigarrow$, in the sense that there is a map*

$$\varphi : F_A \to (\mathsf{List}(A + A) /\!\!/ \rightsquigarrow) \quad (66)$$

such that $\omega \circ \varphi$ is the identity on F_A.

Proof. For any $x : A + A$, the operation "adding x to a list"

$$(x :: \_) : \mathsf{List}(A + A) \to \mathsf{List}(A + A) \quad (67)$$

can be lifted to a function of type

$$(\mathsf{List}(A + A) /\!\!/ \rightsquigarrow) \to (\mathsf{List}(A + A) /\!\!/ \rightsquigarrow). \quad (68)$$

Moreover, the function (68) is inverse to $(x^{-1} :: \_)$ and thus an equivalence.

Let $\star$ be the unique element of the unit type $\mathbf{1}$. We define the relation $\sim$ on the unit type by $(\star \sim \star) :\equiv A$. Then, $\mathsf{hcolim}(A \rightrightarrows \mathbf{1})$ is by definition the coequaliser $(\mathbf{1} /\!\!/ \sim)$, and F_A is given by $(\iota(\star) = \iota(\star))$. This allows us to define φ using Theorem 9 with the constant family $P :\equiv (\mathsf{List}(A + A) /\!\!/ \rightsquigarrow)$, with the equivalence of the component e given by (68).

A further application of Theorem 9 show that $\omega \circ \varphi$ is pointwise equal to the identity. $\qquad\square$

Proof of Theorem 47. By [27, Thms 7.2.9 and 7.3.12], the statement of the theorem is equivalent to the claim that $\|F_A\|_1$ is a set.

We now consider the following diagram:

$$(69)$$

The dashed map exists by the combination of Theorem 46 (note that we are in the simplified case where the type to be quotiented is a set) together with Lemma 48 (and Lemma 8). By construction, the bottom triangle commutes. The top triangle commutes by Lemma 49.

Therefore, the map $|-|_1$ factors through a set (namely $\mathsf{List}(A + A)/\rightsquigarrow$). This means that $\|F_A\|_1$ is a retract of a set, and therefore itself a set. $\qquad\square$

There are a number of deep problems that we can approach in a similar fashion. Let us look at the following list, where the first question is the one discussed in the introduction:

(i) Is the free higher group on a set again a set?
(ii) Is the suspension of a set a 1-type (open problem recorded in [27, Ex 8.2])?
(iii) Given a 1-type B with a base point $b_0 : B$. If we add a single loop around b_0, it the type still a 1-type?
(iv) Given B and b_0 as above, imagine we add M-many loops around b_0 for some given set M. Is the resulting type still a 1-type?
(v) If we add a path (not necessarily a loop) to a 1-type B, is the result still a 1-type?
(vi) If we add an M-indexed family of paths to a 1-type B (for some set M), is the resulting type still a 1-type?

All questions are of the form:

> "Can a change at level 1 induce a change at level 2 or higher?"

Only (i) seems to be about level 0 and 1, but this is simply because we have taken a loop space. With our Theorem 46, we can show an approximation for each of these questions analogously to Theorem 47. This means that we show:

> "A change at level 1 does not induce a change at level 2 (but we don't know about higher levels)."

Fortunately, we do not have to discuss each problem separately as they have a common generalisation. To see this, assume we have a span $B \xleftarrow{f} A \xrightarrow{g} C$ of types and functions.

We then get the following pushout:

$$
\begin{array}{ccc}
A & \xrightarrow{\ g\ } & C \\
{\scriptstyle f}\downarrow & & \downarrow{\scriptstyle i_1} \\
B & \xrightarrow[\ i_0\ ]{} & B +_A C
\end{array}
\qquad (70)
$$

Assume now that A is a set while B and C are 1-types. The above questions all ask whether $B +_A C$ is a 1-type, under the following additional assumptions:

(i) Assuming that B, C are both the unit type $\mathbf{1}$ and A is $A' + \mathbf{1}$, where A' is the set on which we want the free higher group (this is the usual translation from coequalisers to pushouts);

(ii) Assuming that B and C are both $\mathbf{1}$;

(iii) assuming that A is $\mathbf{1}$ and C is the circle S^1;

(iv) assuming that A is $\mathbf{1}$ and C is $M \times \mathsf{S}^1$;

(v) assuming that A is the 2-element type $\mathbf{2}$ and C is $\mathbf{1}$;

(vi) assuming that A is $M \times \mathbf{2}$ and C is M.

With our tools, we can show an approximation of the general problem (and thus approximations of (i) to (vi)):

Theorem 50. *Given a pushout as in* (70), *if A is a set and B, C are 1-types, then all second homotopy groups of $B +_A C$ are trivial. In other words, $\|B +_A C\|_2$ is a 1-type.*

Proof sketch. The argument is almost completely analogous to the proof of Theorem 47. The main difference is that the type $\mathrm{List}(A + A)$ is not sufficient any more. Instead, we need to be slightly more subtle when we encode the equalities in the pushout. The following construction is due to Favonia and Shulman [14], who use it in their formulation of the *Seifert-van Kampen Theorem.*

Given the square in (70) and $b, b' : B$, we consider the type $L_{b,b'}$ of lists of the form

$$
[b, p_0, x_1, q_1, y_1, p_1, x_2, q_2, y_2, \dots, y_n, p_n, b'] \qquad (71)
$$

where[4]

$$
\begin{array}{ll}
x_i : A & y_i : A \qquad\qquad (72) \\
p_0 : b = f(x_1) & p_n : f(y_n) = b' \qquad (73) \\
p_i : f(y_i) = f(x_{i+1}) & q_i : g(x_i) = g(y_i) \qquad (74)
\end{array}
$$

The corresponding relation is generated by

$$
\begin{aligned}
&[\dots, q_k, y_k, \mathrm{refl}, y_k, q_{k+1}, \dots] \rightsquigarrow [\dots, q_k \cdot q_{k+1}, \dots] \\
&[\dots, p_k, x_k, \mathrm{refl}, x_k, p_{k+1}, \dots] \rightsquigarrow [\dots, p_k \cdot p_{k+1}, \dots]
\end{aligned}
\qquad (75)
$$

The statement of the Seifert-van Kampen Theorem is that the set-quotient $L_{b,b'}/\!\!\rightsquigarrow$ is equivalent to the set-truncated type $\|i_0(b) = i_0(b')\|_0$ of equalities in the pushout. Similarly to $L_{b,b'}$, there are three further types of lists where one or both of the endpoints are in C instead of B. In general, we can define a type of lists $L_{x,x'}$ for $x, x' : B + C$, and the Seifert-van Kampen Theorem states that $L_{x,x'}/\!\!\rightsquigarrow$ is equivalent to $\|i(x) = i(x')\|_0$, with $i : B + C \to B +_A C$ given by (i_0, i_1).

---

[4]We remove the 0-truncations around the path spaces. These are without effect here since B, C are 1-types.

Let us compare the lists of the type (71) with the lists $\mathrm{List}(A + A)$ that we have used before for special case (i). In this special case, the elements p_i and q_i in (71) carry no information, and the right summand of $A' + \mathbf{1}$ replaces the additional choice of "left or right" in $\mathrm{List}(A + A)$. Under this transformation, the relations (3) and (75) correspond to each other.

The proof of Theorem 50 then proceeds as follows. The construction of ω and φ is essentially the same as before, using the version of Theorem 9 for pushouts available in [21]. For the relation (75), we can show the analogous to Lemmas 24 and 48. The analogous to (69) is

$$
\begin{array}{ccc}
i(x) = i(x') & & \\
{\scriptstyle \varphi}\downarrow & \searrow^{|-|_1} & \\
L_{x,x'}/\!\!\rightsquigarrow \xrightarrow{\ \omega\ } i(x) = i(x') \xrightarrow{\ |-|_1\ } \|i(x) = i(x')\|_1 \\
{\scriptstyle |-|_0}\downarrow \qquad\qquad \nearrow \\
L_{x,x'}/\!\!\rightsquigarrow
\end{array}
\qquad (76)
$$

There is a small subtlety: Since A is a set and B, C are 1-types, the type of lists $L_{x,x'}$ is a set. This is important since it allows us (as before) to use the simpler version of Theorem 46. The above diagram shows that $\|i(x) = i(x')\|_1$ is a set. Choosing x and x' to be identical, this means that $\|\Omega(B +_A C, i(x))\|_1$ is a set, which is equivalent to the statement that $\|\Omega^2(B +_A C, i(x))\|_0$ (the second homotopy group) is trivial. It follows by the usual induction principle of the pushout that $\|\Omega^2(B +_A C, z)\|_0$ for arbitrary $z : B +_A C$ is trivial. □

8 On Type Theory in Type Theory

There is a long series of suggestions to formalise dependent type theory inside dependent type theory, starting with Dybjer [10]. Chapman [6] introduced the expression "type theory eats itself" for a formalisation of the type-theoretic syntax, inspired by Danielsson [7] and others. Altenkirch and Kaposi [3] refer to the same concept as "type theory in type theory". Another recent suggestion was made by Escardó and Xu [11]. Abel, Öhmann, and Vezzosi [1] start their paper stating that "type theory should be able to handle its own meta-theory".

All proposals construct, in one way or another, types and type families of contexts, substitutions, types, and terms. They further define convertibility relations which capture the intended equalities, stating for example that $(\lambda x.x)(0)$ is related to 0. It is then natural to quotient the constructed types by the relation. Some authors who use homotopy type theory do this without truncating, which leads to a syntax that is not a set and therefore cannot have decidable

equality. Other authors ensure that the constructed syntax is set-truncated and has decidable equality [16]. However, this approach leads to another problem which was discussed by Shulman [26], Altenkirch and Kaposi [3], Kaposi and Kovács [17], and others in the community. The problem is that we want to ensure that the *standard model* or *meta-circular interpretation* is really a model of the syntax. This means that, among other components, we have to construct a function from the set of contexts to the universe of sets (or types). But this universe is not a set.

If we are happy to build the standard model out of sets (meaning that contexts are sets and types are families of sets), then it is a 1-type. This makes the results of this paper relevant. As noted for example in the Agda implementation by Escardó and Xu [11], all required equations hold *judgmentally* in the standard model. In other words, every $p : a \sim b$ gets interpreted as refl. But if every segment of a cycle gets mapped to refl, one might expect that it is easy to prove that the whole cycle gets mapped to refl as required by Theorem 13. Unfortunately, this is not the case. The first statement is meta-theoretic, while the second is internal. Although trivial for any *concretely given* cycle, it is unclear to us how to directly prove that *all* cycles get mapped to refl.

While Theorem 13 seems to not be useful for this reason, we conjecture that a version of Noetherian cycle induction (Theorem 45) can be used. The relations we need in type theory are Noetherian and confluent, and it is very plausible that confluence cycles can be checked. The main difficulty which needs to be overcome is that the authors [3] do not simply first define types of expressions and then the relation, but both components mutually. It seems necessary to adapt our main result accordingly. We have not worked this out and leave it for the future.

9 Final Remarks

Coherence of structures is a central concept in homotopy type theory. In most cases, the difficult part is to *express* coherence. In the present paper, this part is easy: The coherence in question is already expressed correctly in the fairly simple Theorem 13. Instead, *proving* or *checking* the coherence of data that occurred in natural examples is the tricky part.

It is natural to ask what one can say about the higher homotopy groups of the free higher group over a set. It may very well be possible to show "higher" versions of Theorem 46 which relax the condition of 1-truncatedness if X to, for example, 2-truncatedness. The expectation is that this would require coherence for the proofs that one needs for confluence cycles.

We do not believe that the conditions on A, B, and C in Theorem 50 can be relaxed any further. If A is the non-set S^1 we get a counterexample, since the suspension of the circle is the sphere which does not have trivial higher homotopy groups. If B and C are not required to be 1-truncated, Paolo

Capriotti has pointed out to us that the pushout of the span

$$\|S^2\|_2 \leftarrow \mathbf{1} \rightarrow \|S^2\|_2 \tag{77}$$

is not a 2-type. Thus, Theorem 50 is in some sense maximally general.

Finally, we hope that our main construction has applications outside of type theory. The condition that "every cycle becomes trivial" makes sense in graph rewriting, although (as far as we are aware) it has not been considered in the context of well-founded relations. As Christian Sattler has pointed out, there is a connection to van Kampen pushouts; for example, in the category of sets, a pushout is van Kampen if every cycle is trivial [22, Def 21].

Acknowledgments

References

[1] Andreas Abel, Joakim Öhman, and Andrea Vezzosi. 2017. Decidability of conversion for type theory in type theory. *Proceedings of the acm on programming languages* 2, POPL (2017), 23.

[2] Peter Aczel. 1977. An introduction to inductive definitions. In *Studies in Logic and the Foundations of Mathematics*. Vol. 90. Elsevier, 739–782.

[3] Thorsten Altenkirch and Ambrus Kaposi. 2016. Type Theory in Type Theory Using Quotient Inductive Types, In Principles of Programming Languages (POPL'16). *SIGPLAN Not.* 51, 1, 18–29. https://doi.org/10.1145/2914770.2837638

[4] Steve Awodey and Michael A. Warren. 2009. Homotopy Theoretic Models of Identity Types. *Mathematical Proceedings of the Cambridge Philosophical Society* 146 (2009), 45–55. https://doi.org/10.1017/S0305004108001783

[5] Paolo Capriotti, Nicolai Kraus, and Andrea Vezzosi. 2015. Functions out of Higher Truncations. In *24th EACSL Annual Conference on Computer Science Logic (CSL) 2015) (Leibniz International Proceedings in Informatics (LIPIcs))*, Stephan Kreutzer (Ed.), Vol. 41. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 359–373. https://doi.org/10.4230/LIPIcs.CSL.2015.359

[6] James Chapman. 2009. Type theory should eat itself. *Electronic Notes in Theoretical Computer Science* 228 (2009), 21–36.

[7] Nils Anders Danielsson. 2006. A formalisation of a dependently typed language as an inductive-recursive family. In *International Workshop on Types for Proofs and Programs*. Springer, 93–109.

[8] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. 2015. The Lean Theorem Prover. In *Automated Deduction - CADE-25, 25th International Conference on Automated Deduction*.

[9] Peter Dybjer. 1994. Inductive families. *Formal aspects of computing* 6, 4 (1994), 440–465.

[10] Peter Dybjer. 1995. Internal type theory. In *Types for Proofs and Programs (TYPES) (Lecture Notes in Computer Science)*, Stefano Berardi and Mario Coppo (Eds.), Vol. 1158. Springer-Verlag, 120–134. https://doi.org/10.1007/3-540-61780-9_66

[11] Martín Hötzel Escardó and Chuangjie Xu. 2014. Autophagia. https://www.cs.bham.ac.uk/~mhe/TT-perhaps-eating-itself/TT-perhaps-eating-itself.html.

[12] Martin Hofmann. 1995. *Extensional Concepts in Intensional Type Theory*. Ph.D. Dissertation. University of Edinburgh.

[13] Martin Hofmann and Thomas Streicher. 1994. The Groupoid Model Refutes Uniqueness of Identity Proofs. In *LICS*. IEEE Computer Society, 208–212. http://dblp.uni-trier.de/db/conf/lics/lics94.html#HofmannS94

[14] Kuen-Bang Hou (Favonia) and Michael Shulman. 2016. The Seifert-van Kampen Theorem in Homotopy Type Theory. In *25th EACSL Annual Conference on Computer Science Logic (CSL 2016) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 62. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 22:1–22:16. https://doi.org/10.4230/LIPIcs.CSL.2016.22

[15] Gérard P. Huet. 1980. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *J. ACM* 27 (1980), 797–821.

[16] Ambrus Kaposi and Thorsten Altenkirch. 2017. Normalisation by Evaluation for Type Theory, in Type Theory. *Logical Methods in Computer Science* 13 (2017).

[17] Ambrus Kaposi and András Kovács. 2019. Signatures and Induction Principles for Higher Inductive-Inductive Types. *arXiv preprint arXiv:1902.00297* (2019).

[18] Ambrus Kaposi and Jakob von Raumer. 2019. A Syntax for Mutual Inductive Families. Unpublished note.

[19] Chris Kapulkin and Peter LeFanu Lumsdaine. 2018. The Simplicial Model of Univalent Foundations (after Voevodsky). *ArXiv e-prints* (October 2018). arXiv:1211.2851 [math.LO]

[20] Nicolai Kraus and Thorsten Altenkirch. 2018. Free higher groups in homotopy type theory. In *Symposium on Logic in Computer Science (LICS 2018)*. ACM, 599–608.

[21] Nicolai Kraus and Jakob von Raumer. 2019. Path Spaces of Higher Inductive Types in Homotopy Type Theory. In *Symposium on Logic in Computer Science (LICS 2019)*. https://arxiv.org/abs/1901.06022

[22] Michael Löwe. 2010. *Van-Kampen pushouts for sets and graphs*. Technical Report. Fachhochschule für die Wirtschaft Hannover.

[23] Peter LeFanu Lumsdaine. 2010. *Higher Categories from Type Theories*. Ph.D. Dissertation. Carnegie Mellon University. www.andrew.cmu.edu/user/awodey/students/lumsdaine.pdf

[24] Maxwell Herman Alexander Newman. 1942. On theories with a combinatorial definition of" equivalence". *Annals of mathematics* (1942), 223–243.

[25] Tobias Nipkow. 1998. An inductive proof of the wellfoundedness of the multiset order. (1998). https://www21.in.tum.de/~nipkow/misc/ Unpublished note.

[26] Michael Shulman. 2014. Homotopy Type Theory should eat itself (but so far, it's too big to swallow). Blog post, homotopytypetheory.org/2014/03/03/hott-should-eat-itself.

[27] The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book/.

[28] Benno van den Berg and Richard Garner. 2011. Types are Weak ω-Groupoids. *Proceedings of the London Mathematical Society* 102, 2 (2011), 370–394. https://doi.org/10.1112/plms/pdq026 arXiv:http://plms.oxfordjournals.org/content/102/2/370.full.pdf+html

[29] Floris van Doorn, Jakob von Raumer, and Ulrik Buchholtz. 2017. Homotopy type theory in Lean. In *International Conference on Interactive Theorem Proving*. Springer, 479–495. https://arxiv.org/abs/1704.06781