TAL TECH

DOCTORAL THESIS

Datatypes with Symmetries in Homotopy Type Theory

Philipp Joram

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY
TALLINN 2026

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
40/2026

Datatypes with Symmetries in Homotopy Type Theory

PHILIPP JORAM

TAL TECH PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

The dissertation was accepted for the defence of the degree of Doctor of Philosophy on 5 June 2026

Supervisor: Niccolò Veltri,
School of Information Technologies,
Tallinn University of Technology,
Tallinn, Estonia

Opponents: Vikraman Choudhury,
University of Strathclyde,
Glasgow, United Kingdom

Nicolai Kraus,
University of Nottingham,
Nottingham, United Kingdom

Defence of the thesis: 19 June 2026, Tallinn

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted for any academic degree elsewhere.

Philipp Joram _____
signature

ISSN: 2585-6898 (paperback)
ISBN: 978-9916-80-514-5 (paperback)
ISSN: 2585-6901 (PDF)
ISBN: 978-9916-80-515-2 (PDF)
DOI: 10.23658/taltech.40/2026
Printed by EVG Print

Joram, P. (2026). *Datatypes with Symmetries in Homotopy Type Theory* [TalTech Press]. https://doi.org/10.23658/taltech.40/2026

TALLINNA TEHNIKA ÜLIKOOL
DOKTORITÖÖ
40/2026

Sümmeetriatega andmetüübid homotoopilises tüübiteoorias

PHILIPP JORAM

TAL TECH
KIRJASTUS

Contents

List of Publications

Chapters 2 and 3, with the exception of § 3.4, of this thesis are based on the text and results of the following publications:

[I] Philipp Joram and Niccolò Veltri. "Constructive Final Semantics of Finite Bags". In: *14th International Conference on Interactive Theorem Proving (ITP'23)*. LIPIcs 268. 2023, 20:1–20:19. DOI: 10.4230/LIPIcs.ITP.2023.20 (cit. on pp. ix, 7, 57, 176).

[II] Philipp Joram and Niccolò Veltri. "Data Types with Symmetries via Action Containers". In: *30th International Conference on Types for Proofs and Programs (TYPES'24)*. LIPIcs 336. 2025, 6:1–6:21. DOI: 10.4230/LIPIcs.TYPES.2024.6 (cit. on pp. ix, 7, 176).

Chapter 4 is based on the author-contributed text of the following preprint, which is under review for a journal:

[III] Philipp Joram and Niccolò Veltri. "Derivatives for Containers in Univalent Foundations". 2025. arXiv: 2512.17484 [cs.LO] (cit. on pp. ix, 7).

Each of publication is supplemented by a sizable collection of machine-checked definitions and proofs, implemented as *Cubical Agda* code. The relation of these to the results as stated in this thesis is detailed in § 1.3.

Author's Contributions to the Publications

My contributions to the publications [I; II; III] were as follows:

[I] The research problem was identified in collaboration with Niccolò Veltri. I proved the main theorems, developed the majority of the Agda code, and presented the results at the corresponding conference.

[II] I identified the research problem, formulated the definitions, proved the theorems, and verified the results in code. The manuscript was prepared in collaboration with Niccolò Veltri.

[III] I was the main author. I identified the research problem, formulated the main definitions, proved the theorems, and verified the results in code.

1 Introduction

Homotopy type theory connects two areas of mathematics that lie, at a glance, rather far apart. *Homotopy theory* is the study of spaces — geometric objects whose properties we would like to understand, up to some notion of *continuous* deformation. Continuity builds on the primitive notion of a *path*: *points* of a space may be connected by paths, paths themselves may be related by surfaces (paths between paths), surfaces by volumes (paths between surfaces), and so on, all of which have to be preserved by continuous operations. This structure is highly algebraic; understanding the properties of a space often means understanding its ∞-*groupoid* of paths.

Type theory, on the other hand, is a system of discrete logical rules. From some of these rules, one forms *types* that represent mathematical problems and statements. From others one derives *terms* that inhabit such types, which we understand as solutions and proofs of problems and statements, respectively. Martin–Löf's *dependent type theory* (MLTT) is particularly expressive; its two basic *type formers*, Σ and Π, have equally valid interpretations as logical quantifiers and as building blocks of hierarchies of mathematical structures. In a dependent type theory, both the statement "there are infinitely many primes" and the concept of a category are types; both the proof of Euclid's theorem and the category of sets are terms. For any two terms of a type, MLTT posits the existence of an *identity type* whose inhabitants we understand as possible identifications of the pair of terms. Identifications of a pair of terms may again be identified, and so on for all higher identifications of identifications. In a plain *intensional* type theory, this higher structure is, a priori, hard to describe, unless one trivializes it by means of additional axioms such as the principle of *uniqueness of identity proofs* (UIP).

In *Homotopy type theory*, one takes a perspective that links these higher structures of spaces and types: types represent spaces, terms are points of a space, and identity types internalize the notion of path spaces. To understand the higher structure of the *universe* (the paradox-free representation of a "type of all types") itself, Voevodsky introduced the *univalence axiom*. It asserts that identification of types is exactly *equivalence* — a notion of well-behaved transformation of

one type into another. As a consequence, it is possible to *prove* that, for many mathematical structures, Martin–Löf's identity types correspond exactly to our intuitive understanding of "sameness": sets are identified by bijections, groups by isomorphisms, categories by equivalences, and so on for many others.

One can postulate the existence of *higher inductive types*: types of which not only their terms are inductively generated, but also terms of their (higher) identity types. Together with univalence, it is now possible to characterize nontrivial identifications in types, or the higher path structure of spaces, depending on the chosen perspective. A space may have nontrivial self-identifications which, by univalence, are exactly its self-equivalences. We call these self-equivalences *symmetries*. Any type-theoretic construction is closed under substitution along identifications, hence necessarily invariant under symmetries.

Type theory excels at describing, in particular, other type systems. In this thesis, we ask the following question: Can we use homotopy type theory — with its "built-in" treatment of path types — to better understand *datatypes with symmetries*?

How to model mathematically what we understand as a "datatype" is a wide topic with a rich history. At least for datatypes in functional programming languages, the *Curry-Howard-Lambek* correspondence suggests understanding types by means of categorical models. To understand *type polymorphism*, for example, Bainbridge et al. have taught us to model parametric datatypes (lists, tuples, etc.) by functors, and polymorphic functions between them by natural transformations [Bai+90].

Abbott's *containers* [Abb03] model a large class of inductively defined datatypes. A container $(S \triangleleft P)$ consists of a collection of *shapes* S, and for each shape $s : S$ a collection of *positions* P_s. We think of a shape $s : S$ as the name of a term constructor of arity P_s. Containers capture the raw data of a *polynomial* endofunctor $[\![S \triangleleft P]\!](X) = \sum_{s:S} X^{P_s}$, which in turn we understand as a model of the datatype given by this container. When interpreted in a suitably structured base category, morphisms of containers represent exactly the natural transformations of polynomial functors [Abb03, Theorem 4.3.3]. Containers are closed under sums, products, certain exponentials, as well as least- and largest fixpoints. As such, containers can represent all *strictly positive types* [Dyb97]. Furthermore, the category of containers is cartesian closed [ALS10], which lets them express a variety of higher-order concepts.

The types described by containers are in some sense *rigid*: ignoring fixpoints, every type is just a "sum of products", and while this is already good for many practical considerations [dVL14], this fails to describe types that have some internal *symmetry* that programs are supposed to preserve. How should one model an unordered collection (a "bag") and polymorphic functions over it? The obvious functor $\mathsf{M}(X) = \sum_{n:\mathbb{N}} X^n/\sim$ of sums of n-tuples modulo permutation does

not arise from a container. *Quotient containers* [Abb+04] are a straightforward generalization of containers that capture such types. Unfortunately, [Abb+04] does not define substitution and fixpoints of quotient containers, leaving it open whether they define a well-behaved system of composable types.

There are numerous ways in which one might model non-wellfounded types with symmetries, for example as quotients of *polynomial-* [ACH19], or *bounded natural functors* [Für+22]. One might also consider approaches which extend the type theory, for example by a primitive notion of corecursion, such as *guarded recursion* [BMV22]. We follow the path which to us seems most natural in *homotopy* type theory: in order to adequately describe the symmetries of an object, one has to leave the discrete world of sets. Symmetries are best described by groups or groupoids — their categorical counterparts. *Fundamental groupoids* help understanding geometric objects [Bro06], and enumerating isomorphisms of finite structures can become manageable by considering *combinatorial species* (i.e. presheaves on the groupoid of finite sets) [Joy81; BLL97] or their generalizations [Fio+07]. Kock argues that the same applies to data types with symmetries [Koc12, §1.4]; the extension of a quotient container is still just an endofunctor of sets, which cannot adequately capture the necessary symmetries and may explain why a description of substitution of quotient containers is elusive. Gylterud introduced *symmetric containers* [Gyl11] which are interpreted as polynomial pseudofunctors of *groupoids*. Instead of understanding symmetry of data as quotients, these containers have a groupoid of shapes whose set of isomorphisms is preserved by the passage to functors. Polynomials are a well-behaved class of endofunctors over any locally cartesian closed 1-category, where they admit many desirable properties [GK12]. We will see that the same applies to polynomials in h-groupoids, and more generally, higher types.

1.1 Overview

We divide our explorations into three chapters. In chapter 2, we study the behaviour of the type of non-wellfounded, finitely-branching, *unordered* trees, by which we mean trees that are identified up to permutation of their branches. This serves as a case study: can we successfully find a non-trivial (largest) fixpoint type constructively, using the tools provided by homotopy type theory?

Traditionally, such unordered trees are obtained as the largest fixpoint of the endofunctor of *multisets*, i.e. a final coalgebra. Multisets describe the branching of such trees and are usually defined in terms of quotients of sets. In homotopy type theory, we can define a higher inductive type of *set quotients*, which has many desirable properties that are otherwise hard to come by in a plain type theory: induction on quotients matches mathematical intuition, and it is sound to assume that their computation rules hold definitionally, which simplifies proofs

significantly. Furthermore, identity types of quotients by equivalence relations represent exactly equivalence classes. As such there is no need to approximate them by setoids or other ad-hoc means; any construction on quotients necessarily respects the equivalence.

We perform a limit-construction on the multiset functor which, classically, yields its final coalgebra, hence a well-behaved type of non-wellfounded, finitely-branching, unordered trees. We show that, even with the well-behaved HIT of set quotients, this does not happen constructively without the assumption of additional axioms: We construct the desired coalgebra, but prove that it is final if and only if the *lesser limited principle of omniscience* (LLPO) holds — a principle not derivable from our intuitionistic axioms alone, which is verified by some, but not all models of (homotopy) type theory. This issue arises precisely because identity types of set quotients *truncate* the information by which permutation subtrees are identified, and we have to invoke LLPO to recover these data. Instead, we give an alternative presentation of finite multisets as a *homotopy groupoid*: this presentation preserves the data of permutations, and most importantly, lets us encode the branching of trees as a *polynomial* on homotopy groupoids. By a general result of Ahrens et al. [ACS15], such polynomials always admit a final coalgebra in a suitable higher-categorical sense. We ensure that, if set-truncated, this groupoid-based definition is exactly the set-endofunctor of finite multiset. We can show that the truncation of its limit induces a fixpoint in sets without any additional assumptions; proving that it is the largest among such fixpoints, however, so far only seems possible when assuming the axiom of choice.

In chapter 3, we investigate the relationship between such set- and groupoid-based functors more closely. We recognize both finite multisets and their analogue in groupoids are represented by *containers* — the former by a *quotient container* in the sense of Abbott et al. [Abb+04], the latter by one of Gylterud's *symmetric containers* [Gyl11]. We generalize quotient containers slightly, and present *action containers*, which translate into symmetric containers. At its core, this translation is based on the idea of a *delooping of a group*. We implement this as a higher inductive type that assigns to any group a pointed, connected homotopy-groupoid. On top of this we give, step-by-step, 2-categories of groups, actions, and finally families of actions. If combined, these naturally equip both action- and symmetric containers with structures of 2-categories, and thus we can give 2-categorical semantics to our translation: there is a well-behaved 2-functor mapping not only action- to symmetric containers, but also their morphisms and proof-relevant relations of their morphisms. We investigate the apparent lack of a well-defined substitution for action containers. We show that, as 1-categories, action containers are equivalent to symmetric containers equipped with extra structure, and that we cannot expect substitution of symmetric containers to respect this structure for free.

In chapter 4, we fully embrace the higher structure of types and study containers valued in arbitrary types, dropping any truncation-assumptions. The focus of this chapter is the structure of derivatives on type-valued containers. Conceived by McBride for *regular types* [McB01], the derivative computes, for any datatype, a type of *one-hole contexts* of that datatype. Regular types are defined by an inductive grammar (a fragment of strictly positive types), on which their derivative is defined by recursion. In order to describe one-hole contexts adequately, the derivative of sums, products and composition of regular types resembles the rules of its analytic namesake. This lets us mechanically derive notions of "zippers" [Hue97] for a wide variety of datatypes, which are useful, for example, when implementing tree-traversal algorithms.

Abbott et al. extend derivatives to containers [Abb+05]. The derivative of containers is not defined inductively, hence its basic rules no longer hold by definition, but only up to isomorphism of containers. To ensure that the derivative of containers still captures the intuition of a "type with a hole", they establish a universal property characterizing *cartesian* morphisms into derivatives. An "irksome gap" [Abb03, p. 100] of this definition of derivative is that for the universal property to hold, it is necessary to assume that positions of a container are *discrete*, i.e. have decidable identity types. This seemingly restricts derivatives to set-based notions of containers — by *Hedberg's theorem* [Hed98], discrete positions must form homotopy-sets.

We show that, by localizing the derivative to a certain discrete subpart of a container, it is possible to define a derivative for containers valued in arbitrary higher types. This notion of derivative is independent of the truncation level of shapes, and as such lets us study derivatives for datatypes with symmetries in the form of symmetric containers. This generalized derivative still interacts nicely with the remaining structure of containers, and in particular satisfies rules for constants, sums and products. The derivatives of a substitution of containers is subtle, and in general only leads to a *lax chain rule* expressing a sub-container relationship. We derive a rule characterizing derivatives of least fixpoints from the chain rule, which is similarly *lax*. We prove that assuming a strong chain rule (which strengthens the sub-container relationship to an identification) is inconsistent for arbitrary containers, and implies that all homotopy sets are discrete if assumed for set-truncated containers.

1.2 Contributions

Chapter 2. We prove that the classical construction of final coalgebras of set-endofunctors via ω-chains is constructively taboo. For a functor as simple as finite multisets, existence of a final coalgebra is equivalent to LLPO (theorem 2.2.15). We give an alternative definition as a polynomial in groupoids which always

induces a fixpoint of finite multisets if set-truncated (theorem 2.3.12). This is the greatest fixpoint assuming choice (theorem 2.3.13).

Chapter 3. We show that any quotient container gives rise to a symmetric container in a straightforward manner (definition 3.1.21), and that their extensions as endofunctors of sets coincide (theorem 3.1.25). This does not extend to morphisms of quotient containers, which lack certain coherence-properties. We instead introduce action containers (§ 3.2), show how they naturally assemble into a 2-category (§ 3.3.3). We give 2-categorical semantics by constructing a locally fully-faithful 2-functor into symmetric containers (theorem 3.3.31). Alternatively, we give a 1-categorical interpretation by showing that action containers and symmetric containers with the additional structure of a *skeleton* are equivalent as 1-categories (theorem 3.4.17). Through this equivalence we see why a general notion of substitution of set-based containers with symmetries seems to be missing (§ 3.4.3), even if we can prove that substitution is well-defined for certain action containers (corollaries 3.4.23 and 3.4.25).

Chapter 4. We give a construction of a well-behaved derivative operation ∂ on Type-valued containers. This defines a functor on the wild category of Type-valued containers and cartesian morphisms (problem 4.5.8). It extends the derivative for discrete containers of [Abb+05] to an endofunctor on set-truncated containers. The derivative is universal; we construct a wild adjunction $\_ \otimes \mathsf{Id} \dashv \partial$, which characterizes cartesian morphisms mapping into a derivative (problem 4.5.11), and again restricts to set-truncated containers (theorem 4.5.12). We verify the basic properties of a derivative (§ 4.5.1), but observe that the chain rule is not necessarily invertible (problem 4.5.17). Invertibility of the chain rule is inconsistent if assumed for arbitrary Type-valued containers (theorem 4.5.25), and equivalent to decidability of equality for all h-sets if restricted to set-truncated containers (corollary 4.5.27). To establish these *no-go* theorems, we have to study the isolated points of a type (§ 4.4), in particular their behaviour in Σ-types (theorem 4.4.15). Lastly, we describe the derivative of least container-fixpoints (problem 4.5.34), and reduce invertibility of the relevant rule to that of the chain rule (theorem 4.5.40). In order to do so we have to ensure that the traditional construction of least fixpoints of indexed containers [Alt+15] is valid without the assumption of UIP, and yields an initial $F[\_]$-algebra (theorem 4.3.8).

1.3 Notes on formalization

The great majority of results of this thesis have been implemented in the proof assistant *Agda* [Agd05]. Our implementation relies on Agda's built-in support for *cubical type theory* [VMA21], which lets us, among other things, declare higher

inductive types just like any other inductive type, and derive induction principles for which computation rules hold judgementally. Apart from that, we strive to be conservative in the use of the (many) features of Agda; our developments pass the type-checker in `--safe`-mode wherever feasible. Both [Ia] and [IIIa] are fully `--safe`. We hold [IIa] to the same standard, but `postulate` some 2-categorical primitives to ease formalization.[1]

Chapters 2, 3 and 4 of this thesis are formalized, in order, in the following projects:

[Ia] Philipp Joram and Niccolò Veltri. *Constructive Final Semantics of Finite Bags. Agda Formalization*. 2023. Software Heritage Archive: swh:1:dir:7c72e20da77d6dfb3d4a6b8c5221f37709cff68d. URL: https://github.com/phijor/agda-cubical-multiset (cit. on pp. 7, 47, 51, 55).

[IIa] Philipp Joram. *Data Types with Symmetries via Action Containers. Agda Formalization*. 2025. Software Heritage Archive: swh:1:dir:a77c8b642661074b9804670b14cf97cc844aff16. URL: https://github.com/phijor/cubical-containers (cit. on pp. 7, 29, 31, 73, 103).

[IIIa] Philipp Joram. *Derivatives for Containers in Univalent Foundations. Agda Formalization*. 2025. Software Heritage Archive: swh:1:dir:9d6bf6e93cc8afa9d9a2cf1f051d3b1e2e2eb4c9. URL: https://codeberg.org/phijor/derivatives (cit. on pp. 7, 127).

Each project comes with a `README` giving instructions for cross-referencing results between the formalization and [I; II; III]. For convenience, the Agda code is browsable online at the following locations:

- *Constructive Final Semantics of Finite Bags*:

  https://phijor.me/agda-cubical-multiset/README.html

- *Data Types with Symmetries via Action Containers*:

  https://phijor.me/cubical-containers/README.html

- *Derivatives for Containers in Univalent Foundations*:

  https://phijor.me/derivatives/

[1] Specifically, we assume that on 2-cells, composition of 2-functors is unital and associative.

We would like to acknowledge the authors of the `agda/cubical` [Agd25], and the `um-catlab/cubical-categorical-logic` [New+25] libraries, on top of which our development builds. Furthermore, we would like to acknowledge the *1lab* [Lab21], `agda-unimath` [Rij+21], and *TypeTopology* [Ec10] projects, which have been invaluable resources for understanding the intricacies of homotopy type theory both at a technical, as well as an intuitive level.

1.4 Homotopy type theory

In the following sections, we give an overview of the type- and category-theoretic foundations on top of which we build the results of this thesis. We attempt to keep our type-theoretic notation close to that of the HoTT-book [Uni13], our container notation in line with Abbott's original work [Abb03], and our categorical conventions similar to "Displayed Categories" [AL19]. This section is perhaps best read as a reference; skimmed on first read and referenced back for specific definitions and lemmata.

On a meta-mathematical level, we follow Voevodsky's *problem–construction* style. A definition usually introduces a new type. A theorem, proposition, lemma, etc. claims that a type is inhabited. A proof verifies this, but the precise term is usually considered irrelevant. A problem on the other hand asks whether a type has a certain inhabitant. It is a construction that supplies such an inhabitant of interest.

1.4.1 Type-theoretic basics

Judgements, typing, universes. The basic judgement of type theory is $a : A$ — a is a term of type A. All of our types live in a single *universe*, Type, unless stated otherwise. We treat the universe itself as a type, living in a successor universe $\mathsf{Type} : \mathsf{Type}^+$. For $A, B : \mathsf{Type}$, we denote the type of *functions* or *maps* by $A \to B$. Both together let us define families of types $B(a) : \mathsf{Type}$ dependent on $a : A$ as functions $B : A \to \mathsf{Type}$. We denote by $x := y$ a *defining equality* that binds a new name x to a term y. Defining equalities hold *judgementally*, written $x \doteq y$, meaning that we can syntactically replace one with the other, without further justification.

Typeformers. Over a family $B : A \to \mathsf{Type}$, we denote the type of *dependent functions* by $\Pi_A B$ or $\prod_{a:A} B(a)$. Depending on context, we may call $\Pi_A B$ a *dependent product* or a type of *sections* of B. If the type of a can be inferred from $B(a)$, we write $\forall a.\, B(a)$, in particular if B is a kind of predicate on A. A function is introduced by λ-abstraction, we write $(\lambda a.\, b) : \prod_{a:A} B(a)$ where the term b may

reference a. If necessary we add a type annotation to the argument and write $\lambda(a : A).\, b$.

Given $a : A$ and $f : \prod_{a:A} B(a)$, we write function application as $f(a) : B(a)$. For maps in multiple arguments, say $g : \prod_{a:A} B(a) \to C$, we prefer to write $g(a, b) : C$ over $g(a)(b)$. To reduce clutter, we may demote an argument to a subscript and write $g_a(b) : C$ instead. If a is inferable from context, we drop it and write $g(b) : C$. We use an underscore _ to denote *partial application*; $g(a, \_) : B(a) \to C$ is the same term as either g_a or $g(a)$. Composition of functions $f : A \to B$ and $g : B \to C$ is written $g \circ f : A \to C$, or *diagrammatically*, $f ; g : A \to C$. The identity function is $\mathsf{id}_A : A \to A$.

The type of *dependent sums* or *pairs* is denoted by $\Sigma_A B$ or $\sum_{a:A} B(a)$. Given $a : A$ and $b : B(a)$, we introduce a pair by $(a, b) : \sum_{a:A} B(a)$. We denote the two projections from a sum by

$$\mathsf{fst} : \Sigma_A B \to A \qquad \text{and} \qquad \mathsf{snd} : \prod_{x:\Sigma_A B} B(\mathsf{fst}(x))$$

For both Π and Σ-types, we assume the usual β- (computation) and η- (uniqueness) rules judgementally. In particular, we assume $x \doteq (\mathsf{fst}(x), \mathsf{snd}(x))$ for any $x : \Sigma_A B$, although our constructions do not depend on this. We do so in particular in a binder and may write, for example, $\mathsf{snd} : \prod_{(a,b):\Sigma_A B} B(a)$. We write $\Sigma(f, g) : \Sigma_A B \to \Sigma_{A'} B'$ for the map that applies $f : A \to A'$ in the first, and $g : \prod_{a:A} B(a) \to B'(f(a))$ in the second component of a Σ-type.

Inductive types. We write $1 : \mathsf{Type}$ for the unit type with single inhabitant $\bullet : 1$. The empty type is $0 : \mathsf{Type}$; we write $\neg A :\equiv A \to 0$ for constructive negation. Given $A, B : \mathsf{Type}$, their (non-dependent) product $A \times B : \mathsf{Type}$ is the sum over a constant family, i.e. $\sum_{a:A} B$. The coproduct $A + B : \mathsf{Type}$ is generated inductively by the *injections* $\mathsf{inl} : A \to A + B$ and $\mathsf{inr} : B \to A + B$.

We introduce further inductive types by their term-formation rules, i.e. their constructors. The natural numbers $\mathbb{N} : \mathsf{Type}$, for example, come with two constructors

$$\frac{}{\mathsf{zero} : \mathbb{N}} \qquad \text{and} \qquad \frac{n : \mathbb{N}}{\mathsf{suc}(n) : \mathbb{N}}$$

We leave the elimination rules rules implicit and assume the standard computation rules hold judgementally, unless stated otherwise. Instead, we define functions out of inductive types by informally pattern-matching on the argu-

ment:

$$\begin{aligned}&\mathsf{double} : \mathbb{N} \to \mathbb{N}\\&\mathsf{double}(\mathsf{zero}) :\equiv \mathsf{zero}\\&\mathsf{double}(\mathsf{suc}(n)) :\equiv \mathsf{suc}(\mathsf{suc}(\mathsf{double}(n)))\end{aligned}$$

The elimination- and computation rules are sufficient to characterize sections over an inductive type. In the case of natural numbers, for example, we obtain for any family $B : \mathbb{N} \to \mathsf{Type}$ a map

$$\mathsf{elim}_{\mathbb{N}} : B(\mathsf{zero}) \to (\forall n.\, B(n) \to B(\mathsf{suc}(n))) \to (\textstyle\prod_{n:\mathbb{N}} B(n))$$

called its (dependent) *eliminator*. If B is a constant family, we call the resulting map $\mathsf{rec}_{\mathbb{N}} : B \to (\mathbb{N} \to B \to B) \to (\mathbb{N} \to B)$ the *recursor*. We assume that either compute judgementally on constructors, e.g. $\mathsf{elim}_{\mathbb{N}}(z, s, \mathsf{zero}) \doteq z : B(\mathsf{zero})$.

For all $n : \mathbb{N}$, we denote by $[n] : \mathsf{Type}$ the *standard n-element* type. It is defined as an inductive family $[\_] : \mathbb{N} \to \mathsf{Type}$ given by $[\mathsf{zero}] :\equiv 0$ and $[\mathsf{suc}(n)] :\equiv 1 + [n]$.

In prose, we split proofs by induction into cases. For example, in a proof of $\forall(x : A + B).\, P(x)$, we consider the cases $x \doteq \mathsf{inl}(a)$ and $x \doteq \mathsf{inr}(b)$, and prove $P(\mathsf{inl}(a))$ and $P(\mathsf{inr}(b))$ separately. In a more formal system, such proofs should be elaborated into applications of elimination rules.

1.4.2 Identity types and univalence

Identity types. The *identity type* of $x, y : A$ is written $x =_A y$, with the subscript A dropped unless ambiguous. Given $p : x =_A y$, we write $p_* : B(x) \to B(y)$ for transport along p in a family $B : A \to \mathsf{Type}$. For $u : B(x)$ and $v : B(y)$, we treat the type of *dependent paths* $u =^B_p v$ as a primitive, equivalent to its definition in terms of transport, $p_*(u) =_{B(y)} v$. If B is understood, we write $u =_p v$. The action of a map $f : \prod_{a:A} B(a)$ on p is $\mathsf{cong}_f(p) : f(a) =^B_p f(b)$, which we sometimes shorten to $f(p) : f(a) =^B_p f(b)$.

In homotopy type theory, identity types can have interesting, proof-relevant structure. In general, identifications $\gamma : p =_{a=b} q$ of identifications $p, q : a = b$ can be non-trivial. In particular, $\mathsf{refl}_a : a = a$ may not be the only term of type $a = a$. Yet, we can define functions out of identity types by *path induction*, in analogy with the homotopical interpretation of $x = y$ as the type of *paths* from x to y. Let $a : A$ and $P : \prod_{b:A} a = b \to \mathsf{Type}$ a family indexed by paths. In order to define a section $p : \prod_{b:A} \prod_{p:a=b} P(b, p)$, it suffices to give some $r : P(a, \mathsf{refl}_a)$. There exists a function

$$\mathsf{J}_P : P(a, \mathsf{refl}_a) \to \prod_{b:A} \prod_{p:a=b} P(b, p)$$

The computation rule of this J-*rule* asserts that $\mathsf{J}_P(r, a, \mathsf{refl}_a) \doteq r$.

We say that $A : \mathsf{Type}$ is *contractible* if

$$\mathsf{isContr}(A) :\equiv \sum_{a_0:A} \prod_{a:A} a = a_0$$

is inhabited. In this case, $a_0 : A$ is the *center of contraction*. In such a type, any term is (uniquely) identified with any other term. Contractible types encode a notion of uniqueness that is respected by the theory. We understand $\Sigma_A P$ as a proof-relevant existential quantifier; a term $(a, p) : \Sigma_A P$ is evidence that there exists an a that satisfies P, as evidenced by the proof $p : P(a)$. Asserting that $\Sigma_A P$ is contractible is saying that there is such evidence, and that any other proof is uniquely identified with it. Hence, we may write $\exists! a : A. P(a)$ for $\mathsf{isContr}(\sum_{a:A} B(a))$. For any $a : A$, the type of singletons $\mathsf{singl}(a) :\equiv \sum_{a':A} a' = a$ is always contractible; this is called *contractibility of singletons*.

For any given type $A : \mathsf{Type}$, it might be possible to establish its *truncation level*, which measures after which step its tower of identity types trivializes. We say that A is a *proposition* if

$$\mathsf{isProp}(A) :\equiv \prod_{a,b:A} a = b$$

holds. In a proposition, any two terms are identified, but it need not be inhabited. In particular, both 0 and 1 are propositions. One step higher, we have types who themselves might not be trivial, but whose identifications are. We call A a *homotopy set*, or short *h-set*, if there is a proof of

$$\mathsf{isSet}(A) :\equiv \prod_{a,b:A} \mathsf{isProp}(a = b)$$

We will not speak of $p : x =_A y$ as an equality of x and y, unless A is an h-set. We say that $A : \mathsf{Type}$ is *decidable* if $\mathsf{Dec}(A) :\equiv A + \neg A$ is inhabited. The type A is *discrete* if it satisfies $\mathsf{isDiscrete}(A) :\equiv \forall(a, b : A).\ \mathsf{Dec}(a = b)$. *Hedberg's theorem* [Hed98] states that if A is discrete, then A is an h-set.

In general, we say that a type is *n-truncated* or *has truncation level n* if its n-th iteration of path types is trivial. Inductively, we define

$$\begin{aligned}
&\mathsf{isTrunc}_n : \mathsf{Type} \to \mathsf{Type} \\
&\mathsf{isTrunc}_0(A) :\equiv \mathsf{isSet}(A) \\
&\mathsf{isTrunc}_{n+1}(A) :\equiv \prod_{a,b:A} \mathsf{isTrunc}_n(a =_A b)
\end{aligned}$$

Truncation levels are upward-closed; any n-truncated type is also $(n+1)$-truncated.

We extend this hierarchy downwards and say that propositions are (-1)-truncated and contractible types are (-2)-truncated. Unfortunately, there are competing conventions on how to index truncation levels, such as Voevodsky's 0-based indexing [Voe15]. We adopt the (-2)-based convention of the HoTT Book [Uni13]. To avoid confusion, we will however either quantify over all levels, or explicitly use names for low levels: 0-truncated types are *h*-sets, and 1-truncated types are *h-groupoids*. Informally, we will simply speak of *propositions*, *sets* and *groupoids*.

If $P : A \to \mathsf{Type}$ is a family of propositions, then $\Sigma_A P$ defines a *subtype* of A, namely those $a : A$ that satisfy $P(a)$. In particular, $\mathsf{isTrunc}_n(A)$ is a proposition for all n, and we define subtypes of propositions $\mathsf{Prop} := \Sigma_A \mathsf{isProp}$, *h*-sets $\mathsf{hSet} := \Sigma_A \mathsf{isSet}$ and *h*-groupoids $\mathsf{hGrpd} := \Sigma_A \mathsf{isGroupoid}$. We implicitly coerce terms $a : \Sigma_A P$ to type A and write, for example, $f(a) : B$ instead of $f(\mathsf{fst}(a))$ when applying a function $f : A \to B$.

Equivalences. The *fiber* of a map $f : A \to B$ *over* $b : B$ is the type

$$\mathsf{fiber}_f(b) := \sum_{a:A} f(a) = b$$

We think of fibers as a proof-relevant preimage: $\mathsf{fiber}_f(b)$ it is the collection of all $a : A$ that map to b, up to a *chosen* path $p : f(a) = b$. If B is an *h*-set, then $\mathsf{fiber}_f(b)$ is a subtype of A. This preimage is unique if $\mathsf{fiber}_f(b)$ is contractible. We say that f is an *equivalence* if preimages of all points are unique, that is

$$\mathsf{isEquiv}(f) := \prod_{b:B} \mathsf{isContr}(\mathsf{fiber}_f(b))$$

Written in quantifiers, $\mathsf{isEquiv}(f) \doteq \forall b. \exists! a. f(a) = b$. We denote the induced subtype of functions that are equivalences by $A \simeq B$. Equivalences are closed under composition, and the center of contraction of each fiber defines an inverse $f^{-1} : B \to A$, which itself is an equivalence. By contractibility of singletons, $\mathsf{id}_A : A \to A$ is always an equivalence. In general, equivalence is the correct notion of "sameness" of types, which is codified by the univalence axiom.

Univalence. The *univalence axiom* is an extensionality principle for the identity type of the universe. For any $A, B : \mathsf{Type}$, it says that the type $A =_{\mathsf{Type}} B$ is equivalent to $A \simeq B$ in a canonical way. It asserts that, for all $A, B : \mathsf{Type}$, the map

$$\mathsf{idToEquiv} : A = B \to A \simeq B$$

defined by path induction as $\mathsf{idToEquiv}(\mathsf{refl}_A) := \mathsf{id}_A$, is an equivalence itself. We denote its inverse by

$$\mathsf{ua} : A \simeq B \to A = B$$

Univalence implies *function extensionality*; for any $f, g : A \to B$, the types $f = g$ and $\forall a.\, f(a) =_B g(a)$ are equivalent. We say that $p : f = g$ is a *homotopy* of f and g.

1.4.3 Establishing equivalences

To establish equivalences of types, we use a number of strategies. The underlying ideas are not particularly deep, but we use this section as an opportunity to introduce the relevant types and notation.

From isomorphisms. Any equivalence gives rise to a *type isomorphism*. The type of isomorphisms is that of mutually-inverse maps,

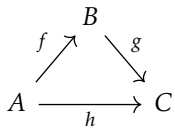
$$(A \cong B) :\equiv \sum_{f:A\to B} \sum_{g:B\to A} (f;g = \mathsf{id}_A) \times (g;f = \mathsf{id}_B)$$

In general, any isomorphism induces some equivalence; there is a map

$$\mathsf{isoToEquiv} : A \cong B \to A \simeq B$$

In practice, this is how we give *some* unspecified equivalence of types A and B. If both A and B are h-sets, then isoToEquiv itself is an equivalence. This not necessarily the case for non-set types.

By decomposition. To show that a specified map $f : A \to B$ is an equivalence, it is often easier to decompose f into a sequence $f = f_1; \ldots; f_n$, and then to show that all f_i are equivalences. Conversely, f itself might be part of a decomposition



In this case we can appeal to the *3-for-2* property: if any two maps in the above diagram are equivalences, then so is the third. From these principles, one can derive a number of techniques that minimize the tears shed over proofs of equivalence [dJon25].

By factorization. In ordinary set theory, a function is a bijection if and only if it is both an injection and a surjection. If phrased correctly, the same holds for equivalences of arbitrary types. We say that $f : A \to B$ is a *surjection* if its fibers are merely inhabited, that is it satisfies

$$\mathsf{isSurj}(f) :\equiv \prod_{y:B} \|\mathsf{fiber}_f(y)\|_{-1}$$

where $\|\_\|_{-1}$ is propositional truncation (see § 1.4.5). The correct notion of injection for types is that of an embedding; f is an embedding if its fibers are propositions, i.e. it satisfies

$$\mathsf{isEmb}(f) :\equiv \prod_{y:B} \mathsf{isProp}(\mathsf{fiber}_f(y))$$

Equivalently, f is a embedding if all fibers over its image are propositions (that is, $\forall x : A.\ \mathsf{isProp}(\mathsf{fiber}_f(f(x)))$), or $\mathsf{cong}_f : x = x' \to f(x) = f(x')$ is an equivalence for all $x, x' : A$. We denote a surjection by $f : A \twoheadrightarrow B$, and an embedding by $f : A \hookrightarrow B$. Together, we obtain the following characterization:

Lemma 1.4.1 ([Uni13, Theorem 4.6.3])**.** A map $f : A \to B$ is an equivalence if and only if it is both an embedding and a surjection. □

In particular, the projection $\mathsf{fst} : \Sigma_A P \to A$ out of a subtype is an embedding. Furthermore, equality of "subtypes" of a chosen type is a proposition:

Lemma 1.4.2. For all $A : \mathsf{Type}$, the type $\mathsf{Emb}(A) :\equiv \sum_{X:\mathsf{Type}} X \hookrightarrow A$ is an h-set. □

In general, we say that a map $f : A \to B$ is *n-truncated* if all of its fibers are n-truncated types. Thus, embeddings are exactly (-1)-truncated maps. The notion of an *embedding of paths* is encoded by *set-truncated maps*:

Lemma 1.4.3 ([Uni13, Lemma 7.6.2])**.** A map $f : A \to B$ is set-truncated (0-truncated) if and only if $\mathsf{cong}_f : x = y \to f(x) = f(y)$ is an embedding ((−1)-truncated) for all $x, y : A$.

The characterization of an equivalence as being both a surjection and an embedding generalizes to it being an n-connected [Uni13, § 7.5] and n-truncated map, which is useful for establishing equivalence of higher types. In general, there are ample *orthogonal factorization systems* [RSS20] that can help construct equivalences.

1.4.4 W-types

In § 4.3, we are going to ensure that indexed containers in Type have least fixpoints. To do so, we need to assume the existence of W-types. Terms of the type $\mathsf{W}(S, P)$ are wellfounded trees with nodes labeled by some $S : \mathsf{Type}$, and a branching given by $P(s) : \mathsf{Type}$ for each label $s : S$. We assume access to W-types in form of the following inductive definition:

Definition 1.4.4. Let $S : \mathsf{Type}$ and $P : S \to \mathsf{Type}$. The inductive type $\mathsf{W}(S,P) : \mathsf{Type}$ is given by the single constructor

$$\frac{s : S \quad f : P(s) \to W(S,P)}{\mathsf{sup}(s,f) : W(S,P)}$$

For $B : \mathsf{W}(S,P) \to \mathsf{Type}$, its eliminator has type

$$\frac{h : \prod_{s:S} \prod_{f:P(s)\to \mathsf{W}(S,P)} (\forall p.\, B(f(p))) \to B(\mathsf{sup}(s,p))}{\mathsf{W\text{-}elim}(h) : \prod_{x:\mathsf{W}(S,P)} B(x)}$$

For non-dependent recursion into some $A : \mathsf{Type}$, we uncurry the recursor:

$$\frac{h : \left(\sum_{s:S} (P(s) \to A)\right) \to A}{\mathsf{W\text{-}rec}(h) : \mathsf{W}(S,P) \to A}$$

On the constructor, W-rec computes to a recursive application of h, namely $\mathsf{W\text{-}rec}(h, \mathsf{sup}(s,f)) \doteq h(s, \mathsf{W\text{-}rec}(h) \circ f)$. ⌟

We think of $\mathsf{W}(S,P)$ as generic inductive type; each label $s : S$ names a constructor with $P(s)$-many recursive occurrences of $\mathsf{W}(S,P)$. W-types have a number of convenient properties.

Lemma 1.4.5. Let $S : \mathsf{Type}$ and $P : S \to \mathsf{Type}$. The following hold:

1. There is an equivalence $\mathsf{W\text{-}in} : \left(\sum_{s:S} (P(s) \to \mathsf{W}(S,P))\right) \simeq \mathsf{W}(S,P)$ induced by the constructor sup.
2. $\mathsf{W}(S,P)$ preserves the truncation level of S, if S is at least a proposition: For all $n \geq -1$, if $\mathsf{isTrunc}_n(S)$, then $\mathsf{isTrunc}_n(\mathsf{W}(S,P))$.
3. $\mathsf{W}(S,P)$ preserves discreteness: If S and $P(s)$ are discrete for all $s : S$, then $\mathsf{W}(S,P)$ is discrete. □

We can recursively describe the fibers of the recursor W-rec:

Problem 1.4.6. Let $S : \mathsf{Type}$, $P : S \to \mathsf{Type}$, $A : \mathsf{Type}$, and $h : (\sum_{s:S} (P(s) \to A)) \to A$. For all $s : S$ and $f : P(s) \to \mathsf{W}(S,P)$, define an equivalence

$$\mathsf{fiber}_{\mathsf{W\text{-}rec}(h)}(\mathsf{W\text{-}rec}(h, \mathsf{sup}(s,f))) \simeq \prod_{p:P(s)} \mathsf{fiber}_{\mathsf{W\text{-}rec}(h)}(\mathsf{W\text{-}rec}(h, f(p)))$$

characterizing the fibers over the image of $\mathsf{W\text{-}rec}(h) : \mathsf{W}(S,P) \to A$.

Construction. Abbreviate $h^* :\equiv \mathsf{W}\text{-rec}(h)$ and let $s : S$, $f : P(s) \to \mathsf{W}(S,P)$. By definition, we have

$$\mathsf{fiber}_{h^*}(h^*(\mathsf{sup}(s,f))) \simeq \sum_{w:\mathsf{W}(S,P)} h^*(w) = h(s, h^* \circ f)$$

Unfolding w via $\mathsf{W}\text{-in} : \sum_{s:S}(P(s) \to \mathsf{W}(S,P)) \simeq \mathsf{W}(S,P)$, we obtain

$$\simeq \sum_{s':S} \sum_{f':P(s')\to\mathsf{W}(S,P)} h^*(\mathsf{sup}(s',f')) = (s, h^* \circ f)$$

By the computation rule for W, this is equivalent to

$$\simeq \sum_{s':S} \sum_{f':P(s')\to\mathsf{W}(S,P)} (s', h^* \circ f') = (s, h^* \circ f)$$

Breaking up the path of pairs into a pair of paths, we rearrange:

$$\simeq \sum_{(s',q):\mathsf{singl}(s)} \sum_{f':P(s')\to\mathsf{W}(S,P)} h^* \circ f' =_q h^* \circ f$$

Contracting $\mathsf{singl}(s)$ and applying function extensionality to the right, we are left with

$$\simeq \sum_{f':P(s)\to\mathsf{W}(S,P)} \prod_{p:P(s)} h^*(f'(p)) = h^*(f'(p))$$

Distributing Σ and Π, we get

$$\simeq \prod_{p:P(s)} \sum_{w:\mathsf{W}(S,P)} h^*(w) = h^*(f(p))$$
$$\simeq \prod_{p:P(s)} \mathsf{fiber}_{h^*}(h^*(f(p)))$$

which finishes the construction of the desired equivalence. ⌟

The above equivalence implies that W-recursion preserves embeddings, which is not immediately obvious:[2]

Lemma 1.4.7. For all S : Type, $P : S \to$ Type, and A : Type, if the function $h : (\sum_{s:S}(P(s) \to A)) \to A$ is an embedding, then so is $\mathsf{W}\text{-rec}(h) : \mathsf{W}(S,P) \to A$.

[2] In fact, we could not find a proof of this anywhere, so we record the full proof for posterity.

Proof. Abbreviate $h^* :\equiv \mathsf{W}\text{-}\mathsf{rec}(h)$. It suffices to show that fibers over the image of h^* are propositions, i.e. $\prod_{w:\mathsf{W}(S,P)} \mathsf{isProp}(\mathsf{fiber}_{h^*}(h^*(w)))$. We do so by W-induction. Let $s : S$, $f : P(s) \to \mathsf{W}(S,P)$. By induction, we can assume

$$\eta : \prod_{p:P(s)} \mathsf{isProp}(\mathsf{fiber}_{h^*}(h^*(f\,p)))$$

By problem 1.4.6, there is an equivalence over $h^*(\mathsf{sup}(s,f))$, namely

$$\mathsf{fiber}_{h^*}(h^*(\mathsf{sup}(s,f))) \simeq \prod_{p:P(s)} \mathsf{fiber}_{h^*}(h^*(f(p)))$$

By the induction hypothesis η, the type on the right is a proposition. Hence, all fibers over the image of h^* are propositions, and h^* is an embedding. □

1.4.5 Higher inductive types

Homotopy type theory lets us introduce inductive types that not only have constructors for terms, but also for (higher) *identifications* involving their terms. Such types are called *higher inductive types* (HITs). The most prominent example of a higher inductive type is the *circle*:

Definition 1.4.8 (The circle)**.** The type S^1 has point- and path constructors

$$\frac{}{\mathsf{base} : S^1} \qquad \text{and} \qquad \frac{}{\mathsf{loop} : \mathsf{base} =_{S^1} \mathsf{base}}$$

Its eliminator is a function

$$\frac{b : B(\mathsf{base}) \quad p : b =^B_{\mathsf{loop}} b}{\mathsf{elim}_{S^1}(b,p) : \prod_{x:S^1} B(x)}$$

for any family $B : S^1 \to \mathsf{Type}$. ⌟

The circle is the simplest example of a type for which UIP fails: S^1 is an h-groupoid, and $\mathsf{loop} \neq \mathsf{refl}$.

Propositional truncation. Perhaps the most frequently used HIT is that of *propositional truncation*:

Definition 1.4.9. For any $A : \mathsf{Type}$, its *propositional truncation* $\|A\|_{-1} : \mathsf{Type}$ is a higher inductive type, generated by a point constructor

$$\frac{a : A}{|a|_{-1} : \|A\|_{-1}}$$

and a path constructor forcing it to be a proposition,

$$\frac{x, y : \|A\|_{-1}}{\mathsf{trunc}(x, y) : x =_{\|A\|_{-1}} y}$$

Its elimination principle characterizes sections of families of propositions. For all $P : A \to \mathsf{Prop}$, there is a map

$$\frac{f^* : \prod_{a:A} P(|a|_{-1})}{\mathsf{elim}_{\|\_\|_{-1}}(f^*) : \prod_{x:\|A\|_{-1}} P(x)}$$

For constant families, this reduces to a recursor $\mathsf{rec}_{\|\_\|_{-1}} : (A \to P) \to (\|A\|_{-1} \to P)$. On the constructor $|\_|_{-1}$, both eliminator and recursor compute to their argument judgementally. ⌟

We understand a term $x : \|A\|_{-1}$ as a proof that A is inhabited by some element, without revealing what that element is — the only way in which we can use x is to inhabit propositions, i.e. proof-irrelevant statements. We say that A is *merely inhabited* if $\|A\|_{-1}$ is inhabited. For example, a map is a surjection if, by definition, all of its fibers are *merely inhabited*. If we can construct a term of

$$\exists a : A.\, P(a) :\equiv \Big\| \sum_{a:A} P(a) \Big\|_{-1}$$

we say that there *merely exists* $a : A$ such that $P(a)$.

We use propositional truncation to hide certain information. Following [Fru+18, § 4.2], for example, we say that a type is a finite set if it is *merely equivalent* to some standard finite set $[n]$:

Definition 1.4.10. A type $A : \mathsf{Type}$ is a *(Bishop) finite set* if it satisfies

$$\mathsf{isFinSet}(A) :\equiv \sum_{n:\mathbb{N}} \|A \simeq [n]\|_{-1}$$

The type of finite sets is $\mathsf{FinSet} :\equiv \sum_{A:\mathsf{Type}} \mathsf{isFinSet}(A)$. ⌟

Note that being a finite set is a proposition, which follows from an application of J and injectivity of the family $[\_] : \mathbb{N} \to \mathsf{Type}$ [Fin+21, § 5]. The standard finite set $[n]$ has decidable equality for any n, which is a proposition. By transport along the mere equivalence $\|A \simeq [n]\|_{-1}$, any finite set A with cardinality n thus enjoys decidable equality, and is necessarily an h-set.

In general, it is not possible to define a section $\prod_{x:\|A\|_{-1}} B(x)$ for an arbitrary family $B : \|A\|_{-1} \to \mathsf{Type}$, unless we know more about B, or can somehow factor it through a proposition. The following lemma lets us perform an elimination into h-sets, provided that we do so in a *weakly constant* manner:

Lemma 1.4.11 ([Kra15, Proposition 2]). For all $A : \mathsf{Type}$ and $B : \mathsf{hSet}$, there is an equivalence of types

$$(\|A\|_{-1} \to B) \simeq \sum_{f:A\to B} \prod_{a,a':A} f(a) = f(a')$$

Any $f : A \to B$ such that $f(a) = f(a')$ for all $a, a' : A$ is called *weakly constant*. □

Set truncation- and quotients. One level up sits the set truncation:

Definition 1.4.12. For any $A : \mathsf{Type}$, its *set truncation* $\|A\|_0 : \mathsf{Type}$ is the HIT generated by constructors

$$\frac{a : A}{|a|_0 : \|A\|_0} \qquad \text{and} \qquad \frac{x, y : \|A\|_0 \quad p, q : x = y}{\mathsf{trunc}(p, q) : p = q}$$

which force it to be an h-set. The eliminator is defined for families $B : A \to \mathsf{hSet}$,

$$\frac{f^* : \prod_{a:A} B(|a|_0)}{\mathsf{elim}_{\|\_\|_0}(f^*) : \prod_{x:\|A\|_0} B(x)}$$

The recursor is typed $\mathsf{rec}_{\|\_\|_0} : (A \to B) \to (\|A\|_0 \to B)$. ⌟

Closely related is the notion of *set quotients*. We call a family $R : A \to A \to \mathsf{Type}$ a *relation* on A. We can define the quotient of A by R as a HIT:

Definition 1.4.13. For any relation R on $A : \mathsf{Type}$, the *set quotient* $A / R : \mathsf{Type}$ is defined inductively by constructors

$$\frac{a : A}{[a] : A/R} \qquad \frac{a, b : A \quad r : R(a, b)}{\mathsf{eq}_{a,b}(r) : [a] =_{A/R} [b]} \qquad \frac{x, y : A \quad p, q : x = y}{\mathsf{trunc}(p, q) : p = q}$$

For families $B : A \to \mathsf{hSet}$, the eliminator is

$$\frac{f : \prod_{a:A} B([a]) \quad h : \prod_{a,b:A} \prod_{r:R(a,b)} f(a) =^B_{\mathsf{eq}(r)} f(b)}{\mathsf{elim}_{A/R}(f, h) : \prod_{x:A/R} B(x)}$$

Recursion is typed $\mathsf{rec}_{A/R} : \prod_{f:A\to B}(\forall a, b.\, R(a, b) \to f(a) = f(b)) \to (A / R \to B)$. ⌟

Intuitively, a function $f : A \to B$ extends to a map $f^* : A / R \to B$ by recursion if it respects the relation R. When giving maps out of a set quotient we will often just set $f^*([a]) :\equiv f(a)$, and ensure that this is *well-defined*, i.e. f respects the relation.

Many higher inductive types can be reduced to some base class of generating HITs. For example, higher truncations exist if the universe is closed under *graph quotients* [Rij17], finitary set-truncated HITs can be constructed from set quotients and propositional truncation [vdWG19], and a large class of h-groupoid truncated HITs are derivable just from a HIT of h-groupoid quotients [VvdW21].

1.4.6 Connected types

We use higher inductive types to express homotopical concepts, such as connectedness: Topologically, a space is *path-connected* if any pair of points is connected by some path. A naive translation of this into $\forall x, y : X.\, x = y$ asserts that X is a proposition, which excludes the majority of types which we would like to consider connected. Instead, we define:

Definition 1.4.14. A type A is connected if it satisfies any of the following equivalent properties:

1. Every pair of points is *merely* connected by path:

$$\prod_{a,b:A} \|a = b\|_{-1}$$

2. A consists of a single *connected component*:

$$\mathsf{isContr}(\|A\|_0)$$

We write $\mathsf{isConnected}(A)$ in either case. ⌟

A straightforward argument by induction shows that S^1, for example, is a connected type in the sense of the first definition. By analogy with the second definition, we understand $\|A\|_0$ as the *set of connected components* of A. By this analogy we have the following decomposition of any type into a sum indexed by its connected components:

Proposition 1.4.15. For all $A : \mathsf{Type}$ there is an equivalence $A \simeq \sum_{y:\|A\|_0} \mathsf{fiber}_{|\_|_0}(y)$ in which $\|A\|_0$ is an h-set, and $\mathsf{fiber}_{|\_|_0}(y)$ is a connected type.

Proof. The equivalence follows from contractibility of singletons. $\|A\|_0$ is an h-set by definition. We show that $\|\mathsf{fiber}_{|\_|_0}(y)\|_0$ is contractible for any $y : \|A\|_0$. The claim follows by induction on y, and the fact that $(|a|_0 = |b|_0) \simeq \|a =_A b\|_{-1}$ [Uni13, Theorem 7.3.12]. □

1.4.7 Choice principles

Classically, the axiom of choice lets us choose inhabitants of each set in an arbitrary family of non-empty sets. In our constructive setting, the veracity of this statement depends on how we interpret "non-emptiness". We use the truncation operators to express type-theoretic variants of the axiom of choice for h-sets and h-groupoids.

Definition 1.4.16 ([Uni13, Ex. 7.8])**.** For $(n,k) \doteq (0,-1)$ or $(n,k) \doteq (1,0)$, define the type

$$\mathsf{AC}_{n,k} :\equiv \prod_{X:\mathsf{hSet}} \prod_{Y:X\to\mathsf{Type}} (\forall x.\, \mathsf{isTrunc}_n(Y(x))) \to \prod_{x:X} \|Y(x)\|_k \to \left\|\prod_{x:X} Y(x)\right\|_k$$

We say that $\mathsf{AC}_{n,k}$ *holds* if it is inhabited. ⌟

We call $\mathsf{AC}_{0,\text{-}1}$ the axiom of choice for h-sets. It is equivalent to the classical assumption that all surjections of sets split.

Lemma 1.4.17. The following are equivalent propositions:

1. $\mathsf{AC}_{0,\text{-}1}$ holds.
2. *All surjections of sets split*: For all $A, B : \mathsf{hSet}$ and $f : A \to B$, if f is a surjection, then there merely exists a section to f, that is $\|\sum_{g:B\to A} \forall b.\, f(g(b) = b\|_{\text{-}1}$.

Proof. If f is a surjection, then $\mathsf{AC}_{0,-1}$ applied to the family $F(b) :\equiv \mathsf{fiber}_f(b)$ shows that f merely has a section. Conversely, if $Y : X \to \mathsf{Prop}$ is some family over $X : \mathsf{hSet}$ such that $\forall x. \|Y(x)\|_{\text{-}1}$, then $\mathsf{fst} : \Sigma_X Y \to X$ is a surjection. If it splits, then $\|\sum_{g:X\to\Sigma_X Y} \prod_{x:X} \mathsf{fst}(g(x)) = x\|_{\text{-}1}$ and the equivalent $\|\prod_{x:X} \sum_{x':X} Y(x') \times x = x'\|_{\text{-}1}$ hold. Thus $\|\forall x.\, Y(x)\|_{\text{-}1}$ holds by contraction of the singleton $\sum_{x':X} x = x'$. □

The axioms $\mathsf{AC}_{n,k}$ are in general not derivable. There are, however, specific choices of X and Y for which $\mathsf{AC}_{n,k}(X,Y)$ holds. For example, if we restrict X to be finite, then $\mathsf{AC}_{0,\text{-}1}(X,Y)$ is provable for all families Y. Proving this *axiom of finite choice* amounts to showing that set truncation distributes over finite products:

Proposition 1.4.18. For any $n : \mathbb{N}$ and family $Y : [n] \to \mathsf{Type}$, the canonical map

$$\mathsf{unbox}_n : \left\|\prod_{k:[n]} Y(k)\right\|_0 \to \prod_{k:[n]} \|Y(k)\|_0$$

induced by $\mathsf{unbox}_n|y|_0 :\equiv \lambda k.\, |y(k)|_0$ is an equivalence.

Proof. We define an inverse box_n to unbox_n by induction on $n : \mathbb{N}$. For $n \doteq 0$, $\mathsf{box}_0(y) : \|\prod_{k:[0]} Y(k)\|_0$ is trivially inhabited for any y. In the successor-case, we define box_{n+1} by lifting the derivable "cons" operation

$$Y(0) \to \left(\prod_{k:[n]} Y(k+1)\right) \to \prod_{k:[n+1]} Y(k)$$

to the set truncation. Showing that both maps are inverses of each other follows immediately by induction on the truncation. □

This equivalence lets us define a variant of the induction principle of set truncation in which the motive is indexed by finite collections $[n] \to \|X\|_0$:

Problem 1.4.19. Let $n : \mathbb{N}$, X : Type and fix a family $B : ([n] \to \|X\|_0) \to \mathsf{hSet}$. Define an induction principle for finite collections

$$\frac{c : \prod_{w:\prod_{k:[n]} Y(k)} B(\lambda k.\,|w_k|_0) \quad v : \prod_{k:[n]} \|Y(k)\|_0}{\mathsf{elim}^{\mathsf{fin}}_{\|X\|_0}(c, v) : B(v)}$$

with propositional computation rule

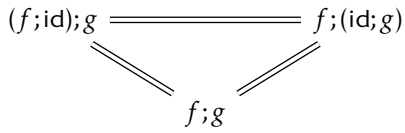
$$\frac{c : \prod_{w:\prod_{k:[n]} Y(k)} B(\lambda k.\,|w_k|_0) \quad w : \prod_{k:[n]} Y(k)}{\mathsf{elim}^{\mathsf{fin}\text{-}\beta}_{\|X\|_0}(c, w) : \mathsf{elim}^{\mathsf{fin}}_{\|X\|_0}(c, \lambda k.\,|w_k|_0) = c(w)}$$

Construction. For appropriately typed c and v, induction on $\mathsf{box}_n(v) : \|\prod_k Y(k)\|_0$ with motive $B \circ \mathsf{unbox}_n$ yields a term $b : B(\mathsf{unbox}_n(\mathsf{box}_n(v)))$. By proposition 1.4.18, unbox_n and box_n cancel, hence we get some $b' : B(v)$ as desired. The computation rule is inherited from the computation rule of $\|\_\|_0$. ⌟

1.5 Category theory

In this thesis, we will verify many desirable properties in the language of category theory. Just like for other concepts we have to take care when translating categorical concepts from set-theoretic foundations into homotopy type theory. Undoubtedly, a category should be two-sorted; there should be a sort of objects, and, to take advantage of the type-theoretic primitives, morphisms should be a sort indexed by a pair of objects. Morphisms compose via a binary operation, which is assumed to be unital and associative.

If we assume that morphisms live in an arbitrary type, then identification of morphisms is not necessarily a proposition. In turn, commutativity of a diagram is structure, not property. We follow [CK17] and call such categories *wild*. These categories are wild in the sense that they come with a choice of paths for unitors ($f ; \mathsf{id} = f$, $\mathsf{id}; f = f$) and associators ($(f ; g); h = f ; (g ; h)$). These paths are structure, and this structure may or may not be coherent; there is no guarantee that, for example, unitors satisfy a triangle identity like



This means in particular that a structure defined in terms of commutativity of diagrams may fail to form a wild category itself. For example, algebras of a wild endofunctor F do not form a wild category unless F is suitably coherent — composition of algebra-morphisms composes commutative squares, which is not guaranteed to be unital or associative.

If the morphism types of a wild category form h-sets, then commutativity of diagrams is a proposition. This is what we shall call a *category*. Nonetheless, the notion of a wild category is useful as an approximation: the universe of types naturally forms a wild category, and maps between types compose as coherently as possible. While we cannot write down the infinite tower of coherences necessary to describe this higher category, we can at least highlight certain well-behaved constructions. For example, we might show that a type of morphisms out of a certain object is contractible, hence showing that this object is universal with respect to some "mapping-out property".

1.5.1 Wild categories

Definition 1.5.1. A wild category $\mathcal{C}$ consists of the following data:

- A type of *objects*, $\mathcal{C}_0 : \mathsf{Type}$.
- A family of morphisms indexed by objects, $\mathcal{C}_1 : \mathcal{C}_0 \to \mathcal{C}_0 \to \mathsf{Type}$.
- An *identity morphism* for each object,

$$\mathsf{id} : \prod_{x:\mathcal{C}_0} \mathcal{C}_1(x,x)$$

- A *composition* operation,

$$(\_;\_) : \prod_{x,y,z:\mathcal{C}_0} \mathcal{C}_1(x,y) \to \mathcal{C}_1(y,z) \to \mathcal{C}_1(x,z)$$

- A choice of left- and right *unitors*, that is families of paths

$$\mathsf{unitl} : \prod_{x,y:\mathcal{C}_0} \prod_{f:\mathcal{C}_1(x,y)} \mathsf{id}_x ; f = f \qquad \text{and} \qquad \mathsf{unitr} : \prod_{x,y:\mathcal{C}_0} \prod_{f:\mathcal{C}_1(x,y)} f ; \mathsf{id}_y = f$$

- A choice of *associators*,

$$\mathsf{assoc} : \prod_{x,y,z,w:\mathcal{C}_0} \prod_{f:\mathcal{C}_1(x,y)} \prod_{g:\mathcal{C}_1(y,z)} \prod_{h:\mathcal{C}_1(z,w)} (f;g);h = f;(g;h)$$

We write $x \to y$ for the type of morphisms $\mathcal{C}_1(x, y)$ if $\mathcal{C}$ is clear from context and there is no risk of confusion with ordinary function types. Composition may be flipped, $g \circ f \coloneqq f ; g$, to mimic function composition. ⌟

The prototypical example of a wild category is that of types and functions. In chapter 4 we are going to study the wild category of Type-valued containers.

Definition 1.5.2. The universe of types, Type, forms the objects of a wild category. Morphisms from A : Type to B : Type are ordinary functions $A \to B$. ⌟

We map between wild categories by means of functors:

Definition 1.5.3. A wild functor $F : \mathcal{C} \to \mathcal{D}$ consists of the following data:

- An action on objects, $F_0 : \mathcal{C}_0 \to \mathcal{D}_0$.
- An action on morphisms, $F_1 : \prod_{x,y:\mathcal{C}_0} \mathcal{C}_1(x, y) \to \mathcal{D}_1(F_0(x), F_0(y))$.
- Two families of chosen paths,

$$\prod_{x:\mathcal{C}_0} F_1(\mathsf{id}_x) = \mathsf{id}_{F_0(x)} \quad \text{and} \quad \prod_{x,y,z:\mathcal{C}_0} \prod_{f:\mathcal{C}_1(x,y)} \prod_{g:\mathcal{C}_1(y,z)} F_1(f ; g) = F_1(f) ; F_1(g),$$

  showing preservation of identities and composition.

⌟

Unlike other concepts, we can express initiality in wild categories coherently as a *property* of an object:

Definition 1.5.4. In a wild category $\mathcal{C}$, and object $0 : \mathcal{C}_0$ is *initial* if for any other $y : \mathcal{C}_0$, the type of morphisms $\mathcal{C}(0, y)$ is *contractible*. ⌟

1.5.2 1-categories

Wild categories become a lot more tame if diagrams commute in at most one way. This is the case when equality of morphisms is a proposition:

Definition 1.5.5. A wild category $\mathcal{C}$ is a *category* if all types of morphisms are h-sets, i.e. it satisfies

$$\mathsf{isCategory}(\mathcal{C}) \coloneqq \prod_{x,y:\mathcal{C}_0} \mathsf{isSet}(\mathcal{C}_1(x, y)) \qquad ⌟$$

Being a category is a proposition, hence categories form a subtype of wild categories. In particular, a functor of categories is just a functor of the underlying wild categories. We compare objects via isomorphisms:

	This thesis	[Uni13]
Morphisms in Type	wild category	—
Morphisms in hSet	category	precategory
idToIso is an equivalence	univalent category	category

Table 1.1: Comparison of terminology between this thesis and the HoTT book.

Definition 1.5.6. In a category $\mathcal{C}$, a morphism $f : \mathcal{C}_1(x,y)$ is an isomorphism if it satisfies

$$\mathsf{isIso}(f) :\equiv \sum_{g:\mathcal{C}_1(y,x)} (f;g = \mathsf{id}_x) \times (g;f = \mathsf{id}_y)$$

The type of all isomorphisms of x and y is $(x \cong_{\mathcal{C}} y) :\equiv \sum_{f:\mathcal{C}_1(x,y)} \mathsf{isIso}(f)$. ⌟

The concept of isomorphism does make sense in a wild category as well, but be aware that there it is genuine structure on morphisms: just as for isomorphisms of types (cf. § 1.4.3), this structure is not necessarily unique if it exists. Isomorphism of objects is supposed to capture equality of objects. Categories in which both concepts coincide are called *univalent*.

Definition 1.5.7. A category $\mathcal{C}$ is *univalent* if for all $x, y : \mathcal{C}_0$, the canonical map

$$\mathsf{idToIso}_{x,y} : x =_{\mathcal{C}_0} y \to x \cong_{\mathcal{C}} y$$

defined by path induction is an equivalence. In this case, isomorphisms form an identity system on objects, and we can use its associated J-rule to perform "isomorphism induction". ⌟

If we restrict types to h-sets, then they form a univalent category.

Example 1.5.8 ([Uni13, Example 9.1.7])**.** The category hSet, whose objects are h-sets and whose morphisms are functions of the underlying types, is univalent. ⌟

Our naming convention for (univalent) categories deviates slightly from the HoTT book.[3] We refer to table 1.1 for a comparison of terminology.

There are many ways to compare categories: isomorphisms, adjunctions, equivalences, and many more. We are mainly going to establish sameness of two categories by means of well-behaved functors:

Definition 1.5.9 ([Uni13, § 9.4])**.** A functor $F : \mathcal{C} \to \mathcal{D}$ of 1-categories is:

1. *fully-faithful* if $F_1 : \mathcal{C}_1(x,y) \to \mathcal{D}_1(F_0(x), F_0(y))$ is an equivalence for all $x, y : \mathcal{C}_0$.

[3] In § 1.5.4 we will also consider two-dimensional notions of categories; our convention will let us avoid clunky names such as *pre-2-category*.

2. *split essentially-surjective* if for all $y : \mathcal{D}_0$ there exists $x : \mathcal{C}_0$ such that $F_0(x) \cong_{\mathcal{D}} y$.

3. *essentially-surjective* if for all $y : \mathcal{D}_0$ there *merely* exists $x : \mathcal{C}_0$ such that $F_0(x) \cong_{\mathcal{D}} y$.

4. an *equivalence* of categories if it is fully-faithful and split essentially-surjective,

5. a *weak equivalence* if it is fully-faithful and essentially-surjective. ⌟

Univalence again lets us characterize identifications, this time of categories:

Proposition 1.5.10 ([Uni13, Lemma 9.4.14])**.** If $\mathcal{C}$ and $\mathcal{D}$ are univalent categories and $F : \mathcal{C} \to \mathcal{D}$ is a weak equivalence, then $\mathcal{C} = \mathcal{D}$ as categories. □

1.5.3 Displayed categories

Many constructions of categories take some existing base category, equip its objects and morphisms with additional structure, and then prove that, together, this new data forms another category. When working type-theoretically, the extra structure on objects of a category $\mathcal{C}$ is usually expressed as a family $D_0 : \mathcal{C}_0 \to \mathsf{Type}$ such that objects of the new category are given by the sum $\sum_{x:\mathcal{C}_0} D_0(x)$, and similarly for morphisms.

Ahrens and Lumsdaine introduce *displayed categories* [AL19], which capture exactly this "extra structure". Over a category $\mathcal{C}$, a displayed category $\mathcal{D}$ is such that together, they form a *total category* $\int_{\mathcal{C}} \mathcal{D}$ whose objects and morphisms are dependent sums of those in $\mathcal{C}$ and $\mathcal{D}$.

Definition 1.5.11. Over a category $\mathcal{C}$, a *displayed category* $\mathcal{D}$ consists of the following data:

- Over each objects of $\mathcal{C}$, a type of *displayed objects*, $\mathcal{D}_0 : \mathcal{C}_0 \to \mathsf{Type}$.
- A family of *displayed morphisms*, indexed by morphisms in $\mathcal{C}$,

$$\mathcal{D}_1 : \prod_{x,y:\mathcal{C}_0} \mathcal{C}_1(x,y) \to \mathcal{D}_0(x) \to \mathcal{D}_1(y) \to \mathsf{hSet}$$

  Given $f : x \to y$ in $\mathcal{C}$ and $a : \mathcal{D}_0(x)$, $b : \mathcal{D}_0(y)$, we write either $\mathcal{D}_1(f; a, b)$ for the set of morphisms over f, or $a \to_f b$ if $\mathcal{D}$ is understood.

- A family of *displayed identity* morphisms, $\mathsf{id} : \prod_{x:\mathcal{C}_0} \prod_{a:\mathcal{D}_0(x)} a \to_{\mathsf{id}_x} a$.

- A *displayed composition*; over $f : \mathcal{C}_0(x,y)$ and $g : \mathcal{C}_0(y,z)$, an operation

$$(\_;\_) : \prod_{a:\mathcal{D}_0(x)} \prod_{b:\mathcal{D}_0(y)} \prod_{c:\mathcal{D}_0(z)} (a \to_f b) \to (b \to_g c) \to (a \to_{f;g} c)$$

- Displayed *left-* and *right-unitors*

$$\mathsf{unitl}' : \prod_{f':a\to_f b} \mathsf{id}_a ; f' =_{\mathsf{unitl}(f)} f' \quad \text{and} \quad \mathsf{unitr}' : \prod_{f':a\to_f b} f' ; \mathsf{id}_b =_{\mathsf{unitr}(f)} f'$$

  for all $f : \mathcal{C}_1(x,y)$, $a : \mathcal{D}_0(x)$ and $b : \mathcal{D}_0(y)$.

- A *displayed associator*

$$\mathsf{assoc}' : \prod_{f':a\to_f b} \prod_{g':b\to_g c} \prod_{h':c\to_h d} (f';g');h' =_{\mathsf{assoc}(f,g,h)} f';(g';h')$$

  for all $f : x \to y$, $g : y \to z$, $h : z \to w$ and $a : \mathcal{D}_0(x)$, $b : \mathcal{D}_1(y)$, $c : \mathcal{D}_1(z)$, $d : \mathcal{D}_1(w)$. ⌟

Most categorical definitions have displayed counterparts, in particular there is a notion of *displayed isomorphism* and *displayed univalence*. The translation of a concept into displayed one is almost mechanical; any quantification of objects is replaced by one of displayed objects over some base object, and similarly for morphisms. Any path is replaced by a dependent path over a its analogue in the base. The *litmus test* for whether a structure S has been translated properly is the following: if $\mathcal{C}$ carries an S-structure, and over it $\mathcal{D}$ carries an analogous displayed S'-structure, then the total category $\int_{\mathcal{C}} \mathcal{D}$ carries an S-structure as well:

Problem 1.5.12. Given a displayed category $\mathcal{D}$ over $\mathcal{C}$, construct the *total category* $\int_{\mathcal{C}} \mathcal{D}$, given by:

- Objects, $\left(\int_{\mathcal{C}} \mathcal{D}\right)_0 \coloneqq \sum_{x:\mathcal{C}_0} \mathcal{D}_0(x)$.
- Morphisms, $\left(\int_{\mathcal{C}} \mathcal{D}\right)_1((x,a),(y,b)) \coloneqq \sum_{f:\mathcal{C}_1(x,y)} \mathcal{D}_1(f;a,b)$.
- Units, $\mathsf{id}_{(x,a)} \coloneqq (\mathsf{id}_x, \mathsf{id}_a)$.
- Composition, $(f,f');(g,g') \coloneqq ((f;g),(f';g'))$.

To highlight the dependency on objects, we sometimes use binders $\int_{x:\mathcal{C}_0} \mathcal{D}_0(x)$ to denote the total category.

Construction. It remains to show that composition in $\int_{\mathcal{C}} \mathcal{D}$ is unital and associative. The displayed unitors and associators in $\mathcal{D}$ are dependent paths over their non-displayed counterparts in $\mathcal{C}$. Together they form exactly the required paths in the Σ-type of morphisms. ⌟

A particularly useful consequence of this is that, in order to establish that a total category is univalent, it suffices to do so separately for the base and its displayed part. In most cases the base will already be univalent, and by an application of isomorphism induction it suffices to show that the fiber categories of a displayed category are univalent:

Definition 1.5.13. Let $\mathcal{D}$ be a category displayed over $\mathcal{C}$. The *fiber category* over $c : \mathcal{C}_0$, $\mathsf{Fib}_{\mathcal{D}}(c)$, consists of the following:

- Objects are those displayed over c,

$$\mathsf{Fib}_{\mathcal{D}}(c)_0 := \mathcal{D}_0(c)$$

- Morphism are $\mathcal{D}$-morphisms displayed over id_c:

$$\mathsf{Fib}_{\mathcal{D}}(c)_1(d, d') := \mathcal{D}_1(\mathsf{id}_c; d, d')$$

Morphisms compose by a restriction of the composition in $\mathcal{D}$. ⌟

Proposition 1.5.14 ([AL19, Theorem 7.4, Proposition 7.3])**.** Let $\mathcal{D}$ be a category displayed over a univalent base $\mathcal{C}$. The total category $\int_{\mathcal{C}} \mathcal{D}$ is univalent if $\mathsf{Fib}_{\mathcal{D}}(c)$ is univalent for all $c : \mathcal{C}_0$. □

1.5.4 2-categories

In § 3.3.3, we are going to compare two notions of container with symmetries. The extra symmetries necessitate studying a proof-relevant relation of container morphisms for either kind of container. This structure is most naturally understood as a two-dimensional category, but tends to be tedious to verify, especially when implemented formally in a dependently typed language. In a univalent setting, Ahrens et al. define *bicategories* and *displayed bicategories* [Ahr+21]. For our purposes the slightly stricter notion of a *2-category* suffices, and we follow [Joh21] in their definition:

Definition 1.5.15. Like a wild category, a *2-category* $\mathcal{C}$ consists of types of objects $\mathcal{C}_0$ and morphisms $\mathcal{C}_1$. In addition, it has an h-set of *2-cells* $\mathcal{C}_2(f, g) : \mathsf{hSet}$ for each parallel pair of morphisms $f, g : \mathcal{C}_1(x, y)$. These compose vertically

$$\_ ;_v \_ : \mathcal{C}_2(f, g) \to \mathcal{C}_2(g, h) \to \mathcal{C}_2(f, h)$$

over $f, g, h : \mathcal{C}_1(x, y)$ and horizontally

$$\_ ;_h \_ : \mathcal{C}_2(f_1, f_2) \to \mathcal{C}_2(g_1, g_2) \to \mathcal{C}_2(f_1 ; g_1, f_2 ; g_2)$$

for $f_1, f_2 : \mathcal{C}_1(x, y)$ and $g_1, g_2 : \mathcal{C}_1(y, z)$. Vertical composition is unital and associative up to a path, with respect to a unit 2-cell $\mathsf{id}_f : \mathcal{C}_1(f, f)$ for all $f : \mathcal{C}_1(x, y)$. Horizontal composition preserves vertical structure (composition and identity),

$$\mathsf{idh} : \mathsf{id}_{f;g} = \mathsf{id}_f ;_h \mathsf{id}_g \qquad \text{and} \qquad \mathsf{exchg} : (s ;_v t) ;_h (u ;_v v) = (s ;_h u) ;_v (t ;_h v)$$

for appropriately indexed 2-cells s, t, u, v. Horizontal composition is unital and associative over the unitors and associators of composition of morphisms: For all $s : \mathcal{C}_2(f_1, f_2)$, $t : \mathcal{C}_2(g_1, g_2)$ and $u : \mathcal{C}_2(k_1, k_2)$, there is some

$$\mathsf{assoch} : (s ;_h t) ;_h u =^{\mathcal{C}_2}_{p_1, p_2} s ;_h (t ;_h u)$$

over $f_i : \mathcal{C}_1(x, y)$, $g_i : \mathcal{C}_1(y, z)$ and $k_i : \mathcal{C}_1(z, w)$, in which $p_i := \mathsf{assoc}(f_i, g_i, k_i)$. Similarly, there is a dependent path over the unitors of morphisms: For any $s : \mathcal{C}_2(f, g)$ over $f, g : \mathcal{C}_1(x, y)$, there are

$$\mathsf{unitlh} : \mathsf{id}_{\mathsf{id}_x} ;_h s =_{\mathsf{unitl}(f)} s \qquad \text{and} \qquad \mathsf{unitrh} : s ;_h \mathsf{id}_{\mathsf{id}_y} =_{\mathsf{unitr}(f)} s \qquad \lrcorner$$

Given objects $x, y : \mathcal{C}_0$, the morphisms in $\mathcal{C}_1(x, y)$ form the objects of a category, called the *local category*. The morphisms of this category are 2-cells with respect to vertical composition. We say that a 2-category is *locally P* if P is a property of categories that holds for *all* local categories. In particular, $\mathcal{C}$ is *locally univalent* if its local categories are univalent, and *locally strict* if each $\mathcal{C}_1(x, y)$ is an h-set [Ahr+21, § 4]. All instances of 2-categories considered in this thesis are either locally strict or -univalent;

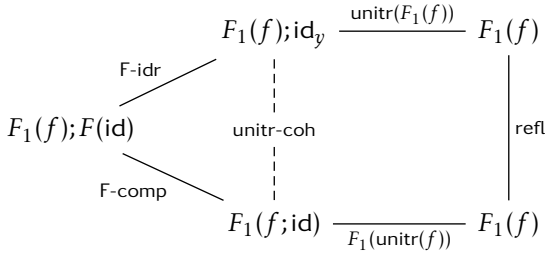
Notice that the horizontal associators and unitors of 2-cells are defined as dependent paths over their 1-dimensional boundaries. This is necessary for the path types to be well-formed: $\mathsf{id} ;_h s : \mathcal{C}_2(\mathsf{id} ; f, \mathsf{id} ; g)$, but $s : \mathcal{C}_2(f ; g)$, and our usage of dependent paths corrects this discrepancy. In general, this lets us avoid the dreaded transport-hell; we prefer using the operations of dependent path-purgatory instead. For 2-categories whose morphisms compose *strictly*,[4] these paths become regular, non-dependent paths.

We define 2-functors of 2-categories to *strictly* preserve the structure of morphisms and 2-cells, i.e. up to a path. For a precise definition, we refer the reader to our formalization[5]. We would like to point out however one particular design

[4] that is, unitors and associators are both refl-paths

[5] IIa, `GpdCont.TwoCategory.StrictFunctor.Base`.

choice. To encode, for example, preservation of unitors of morphisms by a 2-functor $F : \mathcal{C} \to \mathcal{D}$, one has to provide a 2-dimensional path filling the boundary of the pentagon of paths



in which F-idr and F-comp are part of the structure of F as a composition- and identities-preserving map of morphisms. Instead of asking for a filler of the outer diagram directly, we ask for a choice of path $\mathsf{unitr\text{-}coh} : F_1(f);\mathsf{id} = F_1(f;\mathsf{id})$ such that $\mathsf{F\text{-}idr} \cdot \mathsf{F\text{-}comp} = \mathsf{unitr\text{-}coh}$ and the square on the right commutes. The type of fillers

$$\sum_{p:F_1(f);\mathsf{id}=F_1(f;\mathsf{id})} \mathsf{F\text{-}idr} \cdot \mathsf{F\text{-}comp} = p$$

is contractible, hence the type of pentagon-fillers is equivalent to the type of fillers of the right-side square. In practice, finding fillers such squares is considerably easier; in particular when composition of morphisms is strict in both $\mathcal{C}$ and $\mathcal{D}$.

The action of a 2-functor on morphisms and 2-cells defines a functor of local categories, $F_1 : \mathcal{C}_1(x,y) \to \mathcal{D}_1(F_0(x),F_0(y))$. We say that F *locally* has a property if, as functors of 1-categories, its local functors satisfy this property.

The wild category of types (definition 1.5.2) restricts to a 2-category:

Definition 1.5.16. The 2-category hGrpd has as objects h-groupoids and as morphisms maps of their underlying types. Between $f,g : X \to Y$, 2-cells are the h-set of *homotopies* $\mathsf{hGrpd}_2(f,g) :\equiv (f = g)$. These compose vertically and horizontally by path composition. ⌟

This is an instance of the following, more general construction:

Lemma 1.5.17. Any wild category $\mathcal{C}$ whose morphisms form h-groupoids induces a 2-category. The 2-cells are simply identifications of morphisms. □

Thus, we also have a notion of (pseudo-) endofunctor on hGrpd:

Proposition 1.5.18. The 2-category Endo(hGrpd) has as objects wild endofunctors on hGrpd, as morphisms wild natural transformations, and identifications of natural transformations as 2-cells.

Proof. It suffices to ensure that transformations and their naturality squares form an h-groupoid. □

In general, it is possible to identify the exact coherences that a wild category $\mathcal{C}$ has to satisfies such that $\mathsf{Endo}(\mathcal{C})$ forms a 2-category. For our purposes, however, the above example suffices.

For the above definition of 2-category, we derive the appropriate notion of *displayed 2-category*, by a verbose, but mechanical translation. We follow [Ahr+21] which introduces displayed *bicategories*, but adapt to the slightly stricter setting of 2-categories. Again we encode the one- and two-dimensional coherences as paths displayed over their analogues in the base 2-category.[6] For a 2-category $\mathcal{C}$ over which $\mathcal{D}$ is a displayed 2-category, we denote their total 2-category $\int_{\mathcal{C}} \mathcal{D}$. In this 2-category, 2-cells are dependent pairs of 2-cells and displayed 2-cells. In practice, we often define a displayed 2-category $\mathcal{D}$, only to immediately consider $\int_{\mathcal{C}} \mathcal{D}$. In this case, we write $\mathcal{D}$ also for the total category, and disambiguate from context.

By the same pattern, we derive the notion of *displayed 2-functor* $G : \mathcal{D} \to_F \mathcal{D}'$ over some base 2-functor $F : \mathcal{C} \to \mathcal{C}'$. Their *total 2-functor* is denoted by

$$\int_F G : \int_{\mathcal{C}} \mathcal{D} \to \int_{\mathcal{C}'} \mathcal{D}'$$

A displayed functor again induces displayed 1-functors on local categories.

1.6 Containers

Containers were introduced to give categorical semantics to datatypes, in particular the inductively defined class of *strictly-positive types* [Abb03]. A container $(S \lhd P)$ can be thought of a normal form for the datatype it represents; in the set of *shapes* S, each $s \in S$ is understood as the name of a data constructor, and a set of *positions* $P(s)$ records the arguments to this constructor. If $P(s)$ is finite, then one may think of its cardinality as the "arity" of the constructor. Type-theoretically, a container is nothing but a family of positions over a type of shapes:

Definition 1.6.1. A *container* $(S \lhd P)$ consists of *shapes* $S : \mathsf{Type}$ and a family $P : S \to \mathsf{Type}$ of *positions*. We access shapes and positions via postfix projections $(S \lhd P)_{\mathsf{Sh}} :\equiv S$ and $(S \lhd P)_{\mathsf{Ps}} :\equiv P$. ⌟

We will sometimes use binders and write $(s : S \lhd P(s))$ to define containers, just like we do for Σ and Π-types.

To faithfully represent traditional containers, we have to assume that shapes and positions are set-truncated. In general, we define the following subtypes of truncated containers:

[6] IIa, `GpdCont.TwoCategory.Displayed.Base`.

Definition 1.6.2. A container $(S \lhd P)$ is (n,k)-*truncated* if S is n-truncated, and P_s is k-truncated for all $s : S$. We write (n,k)-Cont for the type of (n,k)-truncated containers. ⌟

If n and k coincide, we simply speak of an n-truncated container. In this sense, traditional containers are set-truncated containers, i.e. $(0,0)$-truncated. In a $(1,0)$-truncated container, shapes are 1-types (h-groupoids), and their positions are 0-types (h-sets). In § 3.1.2 we will study such containers; they are Gylterud's *symmetric containers*.

Any container defines a "polynomial" in types, that is a family $\mathsf{Type} \to \mathsf{Type}$, called its *extension*, *extent* or *interpretation*:

Definition 1.6.3. The *extension* of a container $F \doteq (S \lhd P)$ is $[\![F]\!] : \mathsf{Type} \to \mathsf{Type}$ given by $[\![F]\!](X) \coloneqq \sum_{s:S}(P_s \to X)$. ⌟

The extension of set-truncated containers defines an endofunctor of h-sets. A container F represents the polymorphic type (in the sense of [Bai+90]) $[\![F]\!]$. To model (polymorphic) functions between such types, one should describe how constructors of the input map to constructors in the output, and associate to each position in the output — encoding the argument to a constructor — its occurrence in the input.

Definition 1.6.4. Let $F \doteq (S \lhd P)$ and $G \doteq (T \lhd Q)$. The type of *container morphisms* between F and G is

$$(F \to G) \coloneqq \sum_{f:S\to T} \prod_{s:S} Q_{fs} \to P_s$$

We denote the shape- and position components of a morphism $f : F \to G$ by $f_{\mathsf{Sh}} : F_{\mathsf{Sh}} \to G_{\mathsf{Sh}}$ and $f_{\mathsf{Ps}} : \prod_{s:F_{\mathsf{Sh}}} G_{\mathsf{Ps}}(f_{\mathsf{Sh}}(s)) \to F_{\mathsf{Ps}}(s)$, respectively. ⌟

Set-truncated containers and container morphisms form a 1-category, and Abbott et al. prove that this defines a fully-faithful functor into the category of endofunctors. In essence, morphisms of set-truncated containers represent exactly polymorphic functions, that is natural transformations of polynomial endofunctors of sets [AAG05, Theorem 3.4].

In general, a question arises: Should one study the categorical properties of containers, or rather of the functors they represent? For set-truncated containers, which one we pick does not matter; the extent functor defines an equivalence of containers and (a subcategory of) endofunctors. One of our goals is to compare existing variations of containers, and introduce new notions when necessary. For these variations, the correspondence between containers and extensions is not always immediately obvious. For this reason we are going to adopt, informally, and *internal* and *external* perspective: Internally, we study (higher) categories of containers, without making reference to their extensions. Externally, we consider

container functors, establish properties, and if possible, pull them back into containers.

Containers are closed under a number of operations. By recursively mapping to these operations, this is how one computes the normal form of a strictly-positive type as a container.

Definition 1.6.5. The basic operations on containers $F \doteq (S \lhd P)$ and $G \doteq (T \lhd Q)$ are:

- Product:

$$(F \otimes G)_{\mathsf{Sh}} \coloneqq S \times T \qquad\qquad (F \otimes G)_{\mathsf{Ps}} \coloneqq \lambda(s,t).\, P_s + Q_t$$

- Sum:

$$(F \oplus G)_{\mathsf{Sh}} \coloneqq S + T \qquad\qquad (F \oplus G)_{\mathsf{Ps}} \coloneqq \lambda \begin{cases} \mathsf{inl}(s).\, P_s \\ \mathsf{inr}(t).\, Q_t \end{cases}$$

- Substitution:

$$(F[G])_{\mathsf{Sh}} \coloneqq \sum_{s:S} (P_s \to T) \qquad\qquad (F[G])_{\mathsf{Ps}} \coloneqq \lambda(s,f).\, \sum_{p:P_s} Q_{fp}$$

Furthermore, for any $A : \mathsf{Type}$, we define the *constant-* $\mathsf{k}(A) \coloneqq (A \lhd 0)$ and *tuple* containers $\mathsf{y}(A) \coloneqq (1 \lhd A)$. The *identity container* is $\mathsf{Id} \coloneqq (1 \lhd 1)$. ⌟

For set-truncated containers, products and sums form a symmetric monoidal structure with units $\mathsf{k}(1)$ and $\mathsf{k}(0)$. The identity container $\mathsf{Id} \coloneqq (1 \lhd 1)$ is a unit for substitution, which is a non-symmetric monoidal product. The extension functor $[\![\_]\!]$ is monoidal with respect to products, sums and substitution. Furthermore, we have $[\![\mathsf{k}(A)]\!](X) \simeq A$, $[\![\mathsf{y}(A)]\!](X) \simeq (A \to X)$ and $[\![\mathsf{Id}]\!](X) \simeq X$.

We will often implicitly use the following extensionality principle to construct paths between container morphisms:

Lemma 1.6.6 (Extensionality of container morphisms)**.** Let $F, G : \mathsf{Cont}$ and $f, g : F \to G$. There is an equivalence of types

$$(f = g) \simeq \prod_{s:F_{\mathsf{Sh}}} \sum_{p:f_{\mathsf{Sh}}(s)=g_{\mathsf{Sh}}(s)} f_{\mathsf{Ps}}(s) =^B_p g_{\mathsf{Ps}}(s)$$

in which the dependent path varies over the family $B(t) \coloneqq G_{\mathsf{Ps}}(t) \to F_{\mathsf{Ps}}(s)$. □

The above lemma is especially useful whenever shapes of the domain are an inductive type. In this case we can construct identifications of container morphisms by a single induction.

1.7 Groups and actions

In chapter 3, we are going to consider a set-based notion of containers that encodes symmetries by means of a groups acting on positions. Many basic concepts of group theory have a number of equivalent definitions. Let us recall some here, and pick those that translate into a dependently-typed setting with the least friction.

Definition 1.7.1. A *group structure* on an h-set G consists of an identity $1_G : G$, a multiplication $\_\cdot\_ : G \to G \to G$ and an inverse operation $(\_)^{-1} : G \to G$ such that $\_\cdot\_$ is unital with respect to 1_G, associative, and $g \cdot g^{-1} = 1_G = g^{-1} \cdot g$ holds for all $g : G$.

The large type of *groups* is that of all h-sets equipped with a group structure, denoted by $\mathsf{Group} : \mathsf{Type}^+$. ⌟

We implicitly coerce any $G : \mathsf{Group}$ to its underlying type in expressions such as $\prod_{g,h:G} g \cdot h = h \cdot g$. If it becomes necessary to distinguish group structures, we add subscripts $g \cdot_G h$ to operations.

A group homomorphism is a structure-preserving map of groups:

Definition 1.7.2. For all $G, H : \mathsf{Group}$, a map $\varphi : G \to H$ is a *group homomorphism* if it satisfies

$$\mathsf{isGroupHom}(\varphi) :\equiv \prod_{g,g':G} \varphi(g \cdot_G g') = \varphi(g) \cdot_H \varphi(g')$$

Being a group homomorphism is a proposition, and we denote the subtype of homomorphisms by $G \dot{\to} H :\equiv \sum_{\varphi:G\to H} \mathsf{isGroupHom}(\varphi)$. ⌟

We choose the notation $G \dot{\to} H$ (resembling the multiplication $\cdot$ being preserved) since simply writing $G \to H$ does lead to ambiguities, unlike for categories. Any group homomorphism necessarily preserves the identity and inverses of a group. We say that a homomorphism is an *isomorphism*, *equivalence*, *embedding*, etc. if its underlying map of sets is. In particular, we define the type of group-embeddings $G \dot{\hookrightarrow} H :\equiv \sum_{\iota:G\dot{\to}H} \mathsf{isEmb}(\iota)$. Embeddings into a fixed type form an h-set (lemma 1.4.2), hence the same applies to the type of *subgroups* $\mathsf{Sub}(G) :\equiv \sum_{H:\mathsf{Group}} H \dot{\hookrightarrow} G$. Equivalences and isomorphisms of groups are equivalent types, and we prefer the former over the latter.

Using the displayed machinery, turning groups into a category requires minimal effort:

Definition 1.7.3. Group structures form a category displayed over hSet, and we denote the category of groups by $\mathsf{Group} :\equiv \int_{G:\mathsf{hSet}_0} \mathsf{GroupStr}(G)$. ⌟

The unit type 1 has a unique group structure, and we call it the *trivial group*. It is both an initial- and terminal object. Groups are closed under products; for any family $G : J \to \mathsf{Group}$ over $J : \mathsf{hSet}$, the dependent product $\prod_{j:J} G(j)$, together with pointwise multiplication, is the *direct product*, denoted $\Pi_J G : \mathsf{Group}$.

Path composition, if restricted to loops in h-groupoids forms the multiplication of a group:

Definition 1.7.4. Let (X, x_0) be a pointed h-groupoid. The h-set $x_0 = x_0$ is the carrier of the group $\Omega(X, x_0)$, whose identity is refl_{x_0}, multiplication is path composition, and inverse is given by path inversion.

If (X, x_0) is instead a pointed *type*, then its *fundamental group* $\pi_1(X, x_0)$ has carrier $\|x_0 = x_0\|_0$, with operations defined by $\|\_\|_0$-recursion. ⌟

The universe of h-sets itself is a large h-groupoid, so $\Omega(\mathsf{hSet}, X)$ is a (large) group for any $X : \mathsf{hSet}$. Univalence is an identity system on hSet, hence we can replace $\Omega(\mathsf{hSet}, X)$ with a equivalent, but small group:

Definition 1.7.5. The *symmetry group* $\Sigma(X)$ on $X : \mathsf{hSet}$ has as carrier the type of self-equivalences $X \simeq X$. Multiplication is *diagrammatic composition* of equivalences, $e \cdot_{\Sigma(X)} f \coloneqq e; f$. For any $n : \mathbb{N}$, we abbreviate $\Sigma([n])$ by Σ_n. ⌟

In the following, whenever we informally refer to "the symmetries of X" or "permutations of X", we mean $X \simeq X$ and its group structure.

A (right) action of a group G on an h-set is usually encoded as an operation $\_ \triangleleft \_ : X \to G \to X$, satisfying $x \triangleleft 1 = x$ and $(x \triangleleft g) \triangleleft h = x \triangleleft (g \cdot h)$. By currying, it is equivalently a homomorphism into the symmetries of X:

Definition 1.7.6. An *action* of $G : \mathsf{Group}$ on $X : \mathsf{hSet}$ is a group homomorphism $\sigma : G \dot{\to} \Sigma(X)$. ⌟

We choose to encode *right* actions so that multiplication in the group is sent to diagrammatic composition of equivalences, and not functional composition. This choice does not matter, as long as we apply it consistently.

An action $\sigma : G \dot{\to} \Sigma(X)$ might have interesting properties as a group homomorphism. In particular, we say that σ is *faithful* if it is an embedding $G \hookrightarrow \Sigma(X)$. In this case, it exhibits G as a subgroup of permutations of X, and we may call G a *permutation group*.

In any set on which a group acts, we can relate two elements if there is some group element whose action sends one to the other. The equivalences classes of this relation are called orbits:

Definition 1.7.7. The *orbit set* of an action $\sigma : G \dot{\to} \Sigma(X)$ is the set quotient

$$X / \sigma \coloneqq X / \sim_\sigma,$$

defined by the *orbit-relation* $x \sim_\sigma y \coloneqq \exists g : G.\, \sigma_g(x) = y$. ⌟

The actions of a fixed group form a category:

Definition 1.7.8. For any G : Group, the category of G-sets has as objects pairs of sets equipped with an action of G. A morphism $(P, \sigma) \to (Q, \tau)$ is a *G-equivariant* map: a function of sets $f : Q \to P$ (note the direction!) such that

$$\begin{array}{ccc} Q & \xrightarrow{f} & P \\ {\scriptstyle \tau(g)}\downarrow & & \downarrow{\scriptstyle \sigma(g)} \\ Q & \xrightarrow[f]{} & P \end{array}$$

commutes for all $g : G$. ⌟

2 Non-wellfounded, Unordered Trees

In this chapter we perform a case study: Using higher inductive types such as the set-quotient, is it possible to adequately describe the set of finitely- and unorderedly branching, non-wellfounded trees? More specifically, given a endofunctor $\mathsf{M} : \mathsf{hSet} \to \mathsf{hSet}$ taking X to finite multisets $\mathsf{M}(X)$ on X, is it possible to construct a final M-coalgebra? In theorem 2.2.15 we prove that the traditional approach is, somewhat surprisingly, constructively taboo: we construct an M-coalgebra as the limit of an ω-chain which is final if and only if the *lesser limited principle of omniscience* (LLPO), a weak form of the law of the excluded middle, holds.

In § 2.2.3, we attempt to obtain a final coalgebra as a quotient of *ordered trees*, i.e. the constructively non-problematic final coalgebra of the list-functor, yet we encounter the same obstacle. We identify this shortcoming as a limitation inherent to set-based definitions: the type of identifications $xs = ys$ of finite multisets $xs, ys : \mathsf{M}(X)$ records exactly the mere existence of a permutation of xs into ys. The final M-coalgebra, if it exists, lets us equate non-wellfounded binary trees by mere level-wise permutations. These possibly non-wellfounded chains of permutations suffice to decide properties of binary sequences $\mathbb{N} \to 2$, in particular LLPO.

In § 2.3 we define a groupoid-based alternative for finite multisets, called *bags*, in which identifications of bags are exactly an h-set of permutations. In particular, we define $\mathsf{Bag} : \mathsf{hGrpd} \to \mathsf{hGrpd}$ as a polynomial $\mathsf{Bag}(X) = \sum_{b:\mathsf{Bij}} (\mathsf{El}(b) \to X)$, so that we can immediately apply a result by Ahrens et al. which shows that the (wild) category of Bag-coalgebras admits a terminal object [ACS15]. This final coalgebra constructively presents non-wellfounded, finitely branching trees, up to level-wise permutation. In particular, it comes with a *coinduction proof principle*, which lets us establish identifications of trees by means of *bisimulations*.

To verify that this groupoid-based definition of unordered trees corresponds to the set-based one, we prove that the set-truncation of the final Bag-coalgebra is an M-fixpoint (theorem 2.3.12). Assuming the axiom of choice, this is the greatest M-fixpoint (theorem 2.3.13).

2.1 Finite multisets

Finite multisets on a type X can be encoded as a higher inductive type in various ways, three of which we present here. The first is the algebraic presentation of the free commutative monoid, the second as lists modulo permutations, the third as an analytic functor. All definitions are set-based, in the sense that each presentation of finite multisets is, in fact, an h-set. We prove that all three presentations are equivalent as types, and by univalence, propositionally *equal* as endofunctors on hSet (proposition 2.1.14).

Following the work of Choudhury and Fiore, we first present finite multisets algebraically [CF23]. Binary union of finite multisets defines a commutative monoid, universally:

Definition 2.1.1 ([CF23, Definition 2.12])**.** Given $X : \mathsf{Type}$, the *free commutative monoid* on X is the higher inductive type $\mathsf{FCM}(X) : \mathsf{Type}$ generated by the following constructors:

- Point constructors for generators, unit and addition:

$$\frac{x : X}{\eta(x) : \mathsf{FCM}(X)} \qquad \frac{}{\epsilon : \mathsf{FCM}(X)} \qquad \frac{xs, ys : \mathsf{FCM}(X)}{xs \oplus ys : \mathsf{FCM}(X)}$$

- Path constructors for the laws of a commutative monoid:

$$\frac{xs : \mathsf{FCM}(X)}{\mathsf{unitl} : \epsilon \oplus xs = xs} \qquad \frac{xs : \mathsf{FCM}(X)}{\mathsf{unitl} : \epsilon \oplus xs = xs}$$

$$\frac{xs, ys, zs : \mathsf{FCM}(X)}{\mathsf{assoc} : xs \oplus (ys \oplus zs) = (xs \oplus ys) \oplus zs} \qquad \frac{xs, ys : \mathsf{FCM}(X)}{\mathsf{comm} : xs \oplus ys = ys \oplus xs}$$

- Truncation:

$$\frac{}{\mathsf{trunc} : \mathsf{isSet}(\mathsf{FCM}(X))}$$

⌟

We assume a standard eliminator $\mathsf{elim}_{\mathsf{FCM}}$ for this higher inductive type. The corresponding non-dependent recursor characterizes maps out of finite multisets as maps into commutative monoids:

Definition 2.1.2. Let $A : \mathsf{hSet}$ and $f : X \to A$. If A carries the structure of a commutative monoid, i.e. has a unit $0_A : A$ and an addition $\_ + \_ : A \to A \to A$ that are unital, associative and commutative, then f extends to a function

$$\mathsf{rec}^{\mathsf{FCM}}_{(A, 0_A, +)}(f) : \mathsf{FCM}(X) \to A$$

⌟

Note that FCM itself is a commutative monoid, hence this defines an action of FCM on functions:

Definition 2.1.3. Let $f : X \to Y$; define

$$\mathsf{map}_{\mathsf{FCM}}(f) : \mathsf{FCM}(X) \to \mathsf{FCM}(Y)$$
$$\mathsf{map}_{\mathsf{FCM}}(f) \coloneqq \mathsf{rec}^{\mathsf{FCM}}_{(\mathsf{FCM}(Y),\epsilon,\oplus)}(\eta \circ f)$$ ⌟

As given, $\mathsf{FCM}(X)$ is defined for arbitrary $X : \mathsf{Type}$. It is however universal only for h-sets: By induction it is not hard to see that FCM restricts to an endofunctor on hSet, and part of a free-forgetful adjunction between the categories of h-sets and commutative monoids. In fact, we will see that, as a consequence of the theorem 2.1.13, there is a natural equivalence $\mathsf{FCM}(X) \simeq \mathsf{FCM}\|X\|_0$ In this sense FCM is fully specified by its behaviour on h-sets.

In [CF23, § 2.2], finite multisets are alternatively presented as a quotient of list up to swapping of head-elements. For our purposes it will be useful to represent them a list-quotient by a type of permutations, inductively defined from swapping adjacent elements in the middle of a list. We can express this relation of lists directly as an indexed inductive type family:

Definition 2.1.4. The relation $\mathsf{Perm} : \mathsf{List}(X) \to \mathsf{List}(X) \to \mathsf{Type}$ is defined inductively by the following rules:

- Every list is a permutation of itself

$$\frac{xs : \mathsf{List}(X)}{\mathsf{id} : \mathsf{Perm}(xs, xs)}$$

- Any permutation can be extended by swapping adjacent elements:

$$\frac{x, y : X \quad xs, ys, zs : \mathsf{List}(X) \quad p : \mathsf{Perm}(xs +\!\!+ x :: y :: ys, zs)}{\mathsf{swap} : \mathsf{Perm}(xs +\!\!+ y :: x :: ys, zs)}$$ ⌟

In other words, Perm is the reflexive-transitive closure of the relation generated by pairs of lists of the form $[\ldots, x, y, \ldots]$ and $[\ldots, y, x, \ldots]$. This is a very "intensional" way of representing permutations of lists: a proof of $\mathsf{Perm}(xs, ys)$ not only records where each element of xs is moved to in ys, but also how it is moved there. As such, $\mathsf{Perm}(xs, ys)$ is generally not a proposition. Any list $[x, y]$, for example, is related to itself by id and swap'ing twice.

Definition 2.1.5 (PList)**.** For any $A : \mathsf{Type}$, lists *up to permutation* are the type

$$\mathsf{PList}(A) \coloneqq \mathsf{List}(A) \,/\, \mathsf{Perm}$$ ⌟

A second way of defining permutations is via a *relation lifting*, or *relator*: Given a relation R on X, its directed relational lifting is a binary relation on $\mathsf{List}(X)$ that holds whenever *every* element in the first list is related to *some* element in the second list. Recall that *occurrence* of elements in a lists is a proof-relevant predicate where $x \in [] \coloneqq 0$ and $x \in y :: xs \coloneqq (x = y) + x \in xs$. By induction, we can remove an occurrence $m : x \in xs$ from a list $xs : \mathsf{List}(X)$, which yields a new list, $xs \setminus m$.

Definition 2.1.6 (relator)**.** The *directed lifting* of a relation R on X is the relation $\mathsf{DRelator}_R$ on $\mathsf{List}(X)$, generated as follows:

- The empty list is related to any other list:

$$\frac{ys : \mathsf{List}(X)}{\mathsf{nil} : \mathsf{DRelator}_R([], ys)}$$

- If $R(x,y)$ for some $y \in ys$, and xs is related to ys without y, then $x :: xs$ is related to ys:

$$\frac{x : X \quad xs, ys : \mathsf{List}(X) \quad p : \exists y : X.\, R(x,y) \times \sum_{m:(y \in ys)} \mathsf{DRelator}_R(xs, ys \setminus m)}{\mathsf{cons}(x,p) : \mathsf{DRelator}_R(x :: xs, ys)}$$

The *lifting* of R is the symmetrization

$$\mathsf{Relator}_R(xs, ys) \coloneqq \mathsf{DRelator}_R(xs, ys) \times \mathsf{DRelator}_R(ys, xs) \qquad \lrcorner$$

The assignment $\lambda R.\, \mathsf{Relator}_R$ can be extended to a *relator* in the sense of [Lev11, § 3]. It preserves a number of properties of R, in particular:

Lemma 2.1.7. For any relation R on X, $\mathsf{DRelator}_R$ is a propositional relation. It is an equivalence relation whenever R is reflexive and transitive. □

Lists are related by $\mathsf{Relator}_R$ whenever all elements in one list are R-related to all elements in the other. In the case of $R(x,y) \coloneqq (x = y)$, this holds exactly when the lists are merely permutations of each other.

Proposition 2.1.8. For all $xs, ys : \mathsf{List}(X)$, the propositions $\mathsf{Relator}_=(xs, ys)$ and $\|\mathsf{Perm}(xs, ys)\|_{-1}$ are equivalent types. □

This yields a second presentation of finite multisets in terms of a list-quotient.

Definition 2.1.9. For any $X : \mathsf{Type}$, lists *up to relator* are the type

$$\mathsf{RList}(X) \coloneqq \mathsf{List}(X) \,/\, \mathsf{Relator}_= \qquad \lrcorner$$

Corollary 2.1.10. For all $X : \mathsf{Type}$, the types $\mathsf{PList}(X)$ and $\mathsf{RList}(X)$ are equivalent.

Proof. In general, A/R and $A/\lambda x\, y.\, \|R(x,y)\|_{-1}$ are equivalent types for any $A : \mathsf{Type}$ and relation R. Logically equivalent relations induce equivalent quotients, so by proposition 2.1.8 we have

$$\mathsf{List}(X) \,/\, \mathsf{Relator}_{=} \simeq \mathsf{List}(X) \,/\, \|\mathsf{Perm}\|_{-1} \simeq \mathsf{List}(X) \,/\, \mathsf{Perm}$$ □

In both presentations of finite multisets as quotients of lists the relation stratifies lists by length; two related lists necessarily have the same length. We can bake this assumption into the definition, and index finite multisets by their cardinality:

Definition 2.1.11. For any $X : \mathsf{Type}$, define

$$\mathsf{FMSet}(X) :\equiv \sum_{n:\mathbb{N}} ([n] \to X \,/\sim_n)$$

where $\sim_n$ is a relation on $[n] \to X$ given by

$$v \sim_n^{\infty} w :\equiv \textstyle\sum_{\sigma:[n]\simeq[n]} v = w \circ \sigma$$
$$v \sim_n w :\equiv \|v \sim_n^{\infty} w\|_{-1}$$ ⌟

When restricted to h-sets, this presents finite multisets as an *analytic functor* in the sense of [Joy86; Has02] — a functor that is a sum of quotients of representables.

Problem 2.1.12. Extend FMSet to a functor $\mathsf{hSet} \to \mathsf{hSet}$.

Construction. Denote the action on $f : X \to Y$ by $\mathsf{map}_f : \mathsf{FMSet}(X) \to \mathsf{FMSet}(Y)$. It preserves cardinality, and is given by precomposition on representatives, $\mathsf{map}_f(n,[v]) :\equiv (n,[f \circ v])$, which is well-defined. Showing that map preserves identities and composition follows directly by induction on the quotient. ⌟

Since FMSet is defined as a set quotient, it necessarily collapses any higher structure of its argument. In particular, it is invariant under set truncation:

Theorem 2.1.13. For all $X : \mathsf{Type}$, $\mathsf{map}_{|\_|_0} : \mathsf{FMSet}(X) \to \mathsf{FMSet}\|X\|_0$ is an equivalence.

Proof. We use induction on finite collections (problem 1.4.19) to define a function typed $([n] \to \|X\|_0) \to ([n] \to X)/\sim_n$. This is enough to obtain some map $\mathsf{FMSet}\|X\|_0 \to \mathsf{FMSet}\, X$, which is an inverse to $\mathsf{map}_{|\_|_0}$ by the computation rule for induction on finite collections. □

All presentations of finite multisets introduced in this section define equivalent types. Moreover, when restricted to h-sets, they are all propositionally *equal* as endofunctors on hSet.

Proposition 2.1.14. The type constructors FCM, FMSet, PList and RList are naturally isomorphic functors of sets. Since hSet is univalent, they are propositionally equal.

Proof. Equivalence of PList and RList is established in corollary 2.1.10, that this equivalence is natural follows directly. We establish FCM = PList and FCM = FMSet by showing that both satisfy the universal property of FCM, i.e. that both $\mathsf{PList}(A)$ and $\mathsf{FMSet}(A)$ are the free commutative monoid on A. In the first case, addition is lifted from concatenation of lists. Addition on $\mathsf{FMSet}(A)$ adds cardinalities, and merges "vectors" $v : [n] \to X$ and $w : [m] \to X$ into some $[n+m] \to X$ via the equivalence $(([n] \to X) \times ([m] \to X)) \simeq ([n+m] \to X)$. □

In the following constructions, the above lets us choose whatever is the most convenient presentation of finite multisets based on the structure of the problem — univalence guarantees that these construction are independent of the choice of presentation. In particular, we state a number of theorems in terms of some abstract finite multiset functor $\mathsf{M} : \mathsf{hSet} \to \mathsf{hSet}$, but give their proof using a concrete presentation.

2.1.1 Definable quotients and sorting

In order to show that the traditional construction of a final M-coalgebra is inherently non-constructive, we need to pick representatives from equivalence classes of multisets. If a set X is linearly ordered, then sorting multisets is a normalisation function $\mathsf{M}(X) \to \mathsf{List}(X)$ — multisets in the same equivalence class sort to propositionally equal lists. Such a representation of quotients in terms of normalisation functions has been proposed by Li under the name of *definable quotient* [Li15], which we adapt here:

Definition 2.1.15. For A : Type and a binary relation R on A, we say that the quotient A/R carries a *definable* structure if the equivalence class constructor $[\_] : A \to A/R$ admits a section $\mathsf{rep} : A/R \to A$. As a type,

$$\mathsf{Definable}_A(R) :\equiv \sum_{\mathsf{rep}:A/R\to A} \prod_{x:A/R} [\mathsf{rep}(x)] = x$$ ⌟

Requiring the representative-picking map to be a section to [_] ensures that each representative belongs to the class it was picked from. Li's definable quotients [Li15, Definition 4.5] are always assumed to be *complete*: A/R is complete

if any $a : A$ is related to its representative, i.e. $\prod_{a:A} \mathsf{rep}[a]\ R\ a$. Under some mild assumptions, definable quotients as in definition 2.1.15 are always complete.

Proposition 2.1.16. If A/R is a definable quotient and R is effective, i.e. admits a map $\prod_{a,b:A}[a] = [b] \to a\ R\ b$, then A/R is complete. In particular, all propositionally-valued equivalence relations are effective.

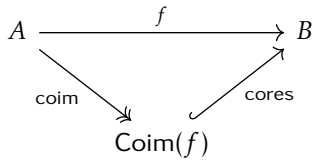
Proof. Let $a : A$. To show $\mathsf{rep}[a]\ R\ a$, it suffices to give some $[\mathsf{rep}[a]] = [a]$, by effectiveness of R. But rep is a section of $[\_]$, hence such a path exists. □

Note that the definable structure on a quotient is not necessarily unique; there can be many distinct sections. Furthermore, existence of a definable structure on arbitrary quotients is a strong assumption, equivalent to the axiom of choice:

Proposition 2.1.17. The following are equivalent propositions:

1. $\mathsf{AC}_{0,\text{-}1}$ of definition 1.4.16 holds.
2. *All surjections of sets split*: For all $A, B : \mathsf{hSet}$ and $f : A \to B$, if f is a surjection, then there merely exists a section to f.
3. *All set-quotients are merely definable*: For all $A : \mathsf{hSet}$ and relations R, there merely exists a definable structure on A/R.

Proof. By lemma 1.4.17, all surjections of sets split if and only iff $\mathsf{AC}_{0,\text{-}1}$ holds. Next, assume that all surjections of sets split. The map $[\_] : A \to A/R$ is a surjection of sets, hence it merely has a section. Lastly, assume that all set-quotients are merely definable. Any function of sets $f : A \to B$ factors (uniquely) through its coimage



which is defined as the quotient $\mathsf{Coim}(f) \coloneqq A/\sim$ where $x \sim y \coloneqq f(x) = f(y)$. The surjection onto Coim is just the equivalence class constructor, i.e. $\mathsf{coim} \doteq [\_]$, hence it merely admits a section $\mathsf{rep} : \mathsf{Coim}(f) \to A$. If f is a surjection, then cores is an equivalence, so $\mathsf{cores}^{-1}; \mathsf{rep} : B \to A$ is a section of f. □

If X is linearly ordered, then *insertion sort* induces a permutation between any input list $xs : \mathsf{List}(X)$ and the sorted output $\mathsf{isort}(xs) : \mathsf{List}(X)$. Of course, any two lists related by a permutation sort to the same output, and this induces a (weakly) constant map on the classes of $\mathsf{List}(X)/\mathsf{Perm}$. Hence, sorting extends to a map $\mathsf{List}(X)/\mathsf{Perm} \to \mathsf{List}(X)$, which we prove to be a section to $[\_]$. Representing $\mathsf{M}(X)$ by $\mathsf{PList}(X)$, shows that it is a definable quotient. Let us make this precise.

Definition 2.1.18. A relation $\_ < \_ : A \to A \to \mathsf{Type}$ on $A : \mathsf{Type}$ is *linear* if it is:

- *propositional*: $x < y$ is a proposition for all $x, y : A$.
- *asymmetric*: For all $x, y : A$, if $x < y$ then $\neg(y < x)$.
- *transitive*: For all $x, y, z : A$, if $x < y$ and $y < z$ then $x < z$.
- *total*: For all $x, y : A$, $(x < y) + (x = y) + (y < x)$ is inhabited.

If A is a set, we call $(A, <)$ a *linearly ordered set*. ⌟

For any linearly ordered set $(A, <)$, we define $\mathsf{insert} : A \to \mathsf{List}(A) \to \mathsf{List}(A)$ such that $\mathsf{insert}(x, ys)$ returns ys with x inserted before the first $y \in ys$ such that $x < y$. Insertion sort, $\mathsf{isort} : \mathsf{List}(X) \to \mathsf{List}(X)$, is then implemented by folding over insert. Sorting is invariant under permutations of the input, and this induces a well-defined normalisation function $\mathsf{PList}(X) \to \mathsf{List}(X)$. For each class it picks a representing list, namely the unique list in ascending order.

Problem 2.1.19. Extend insertion sort on a linearly ordered set $(A, <)$ to a map

$$\mathsf{sort} : \mathsf{List}(A) / \mathsf{Perm} \to \mathsf{List}(A)$$

Construction. Since A is a set, so is $\mathsf{List}(A)$. Thus, we can define sort by recursion on the set quotient. On representatives, let

$$\mathsf{sort}([xs]) := \mathsf{isort}(xs)$$

It remains to show that this is well-defined, i.e. that

$$\prod_{xs, ys : \mathsf{List}(A)} \mathsf{Perm}(xs, ys) \to \mathsf{isort}(xs) = \mathsf{isort}(ys).$$

This follows by induction on the permutation, and inspection of the recursive definitions of isort and insert: For id, the conclusion holds judgmentally; in case of swap, we have to show that

$$\mathsf{isort}(xs +\!\!+ x :: y :: ys) = \mathsf{isort}(xs +\!\!+ y :: x :: ys),$$

which follows by inspection of the trichotomy $(x < y) + (x = y) + (x > y)$. ⌟

Proposition 2.1.20. For a linearly ordered set $(A, <)$, $\mathsf{List}(A) / \mathsf{Perm}$ is a definable quotient with representative-picking map $\mathsf{sort} : \mathsf{List}(A) / \mathsf{Perm} \to \mathsf{List}(A)$.

Proof. We need to show that $[\mathsf{sort}\, xs] = xs$ for all $xs : \mathsf{List}(A)/\mathsf{Perm}$. This is a family of paths in a set quotient, hence a proposition. By induction on $\mathsf{List}(A) / \mathsf{Perm}$ it suffices to show that $[\mathsf{isort}(ys)] = [ys]$ for any $ys : \mathsf{List}(A)$. But $\mathsf{isort}(ys)$ and ys are related by a permutation, hence equal in the quotient. □

This lets us define a linear order on lists up to permutation. Recall that any linear order on a set extends to a linear order on lists.

Definition 2.1.21. Given a linearly ordered set $(A, <)$, the *lexicographic order* $<_{\mathsf{lex}}$ on $\mathsf{List}(A)$ is defined inductively by

$$\frac{}{[] <_{\mathsf{lex}} (y :: ys)} \qquad \frac{x < x'}{(x :: xs) <_{\mathsf{lex}} (x' :: xs)} \qquad \frac{x = y \quad xs <_{\mathsf{lex}} ys}{(x :: xs) <_{\mathsf{lex}} (y :: ys)}$$ ⌟

Sorting gives us representatives for unordered lists. Hence, we compare unordered lists by first sorting them, and then comparing their representatives:

Problem 2.1.22. Given a linearly ordered set $(A, <)$, extend $<_{\mathsf{lex}}$ to a linear order $<_{\mathsf{plex}}$ on $\mathsf{List}(A) / \mathsf{Perm}$.

Construction. For all $xs, ys : \mathsf{List}(A) / \mathsf{Perm}$ define

$$xs <_{\mathsf{plex}} ys := \mathsf{sort}(xs) <_{\mathsf{lex}} \mathsf{sort}(ys)$$

Asymmetry, transitivity and propositionality of this relation are inherited from $<_{\mathsf{lex}}$. Totality follows similarly, except that we have to ensure that for all $xs, ys : \mathsf{PList}(A)$, if $\mathsf{sort}(xs) = \mathsf{sort}(ys)$ then $xs = ys$. But since sort is a section to $[\_]$, we have

$$xs = [\mathsf{sort}(xs)] = [\mathsf{sort}(ys)] = xs,$$

as desired. ⌟

Stratifying these results by cardinality of multisets, we obtain the following structure on "vectors" of type $([n] \to X) / \sim_n$:

Proposition 2.1.23. If $(X, <)$ is a linearly ordered set, then $([n] \to X) / \sim_n$ is

1. a definable quotient,
2. linearly ordered.

Proof. By proposition 2.1.14, $\mathsf{PList}(X) \simeq \mathsf{FMSet}(X) \simeq \sum_{n:\mathbb{N}} ([n] \to X) / \sim_n$. By proposition 2.1.20 and problem 2.1.22, $\mathsf{PList}(X)$ is a definable quotient and linearly ordered via sorting. We transport this structure to $\mathsf{FMSet}(X)$, and observe that it restricts to multisets of the same cardinality. □

Recall that $\sim_n$ hides the permutation by which vectors are related under a propositional truncation. Over linearly ordered sets, we can recover a "canonical permutation" for any two related vectors:

Proposition 2.1.24. If $(X, <)$ is a linearly ordered set, then there exists a function

$$\prod_{n:\mathbb{N}} \prod_{v,w:[n]\to X} (v \sim_n w) \to (v \sim_n^\infty w)$$

induced by map

$$\mathsf{canonPerm} : \prod_{xs,ys:\mathsf{List}(X)} \mathsf{Perm}(xs, ys) \to \mathsf{Perm}(xs, ys)$$

that is *weakly constant*, i.e. satisfies $\mathsf{canonPerm}(\sigma) = \mathsf{canonPerm}(\tau)$ for all $\sigma, \tau : \mathsf{Perm}(xs, ys)$.

Proof. Again, it is enough to define a map $\mathsf{c} : \|\mathsf{Perm}(xs, ys)\|_{-1} \to \mathsf{Perm}(xs, ys)$ for all $xs, ys : \mathsf{List}(X)$ and then to stratify by cardinality. The codomain of c is an h-set, so by lemma 1.4.11, it suffices to give

$$\mathsf{canonPerm} : \mathsf{Perm}(xs, ys) \to \mathsf{Perm}(xs, ys)$$

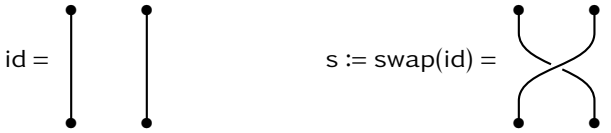
and show that it is weakly constant. Any permutation $\sigma : \mathsf{Perm}(xs, ys)$ yields a path $p_\sigma : \mathsf{sort}(xs) = \mathsf{sort}(ys)$, and a permutation $\bar{\sigma} : \mathsf{Perm}(\mathsf{sort}(xs), \mathsf{sort}(ys))$ by transport along p_σ. Similarly, we have $\mathsf{sortPerm} : \prod_{xs:\mathsf{List}(X)} \mathsf{Perm}(xs, \mathsf{sort}(xs))$. Permutations are closed under composition and inverses, so we define $\mathsf{canonPerm}(\sigma)$ as the conjugation

$$\begin{array}{ccc} xs & \overset{\mathsf{canonPerm}(\sigma)}{\dashrightarrow} & ys \\ {\scriptstyle \mathsf{sortPerm}(xs)}\downarrow & & \uparrow{\scriptstyle \mathsf{sortPerm}(ys)^{-1}} \\ \mathsf{sort}(xs) & \xrightarrow[\bar{\sigma}]{} & \mathsf{sort}(ys) \end{array}$$

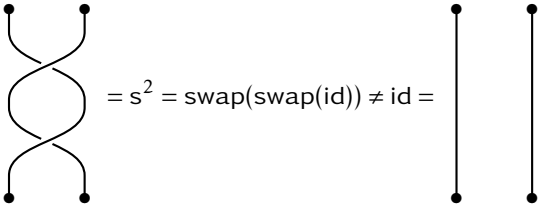
This map is weakly constant because it factors through a proposition: $\mathsf{List}(X)$ is a set, so $p_\sigma = p_\tau$ no matter the choice of σ and τ. □

The computational behaviour of canonPerm depends on the underlying sorting algorithm, and in particular how $\mathsf{sortPerm}(xs) : \mathsf{Perm}(xs, \mathsf{sort}(xs))$ is computed. We implement on top of insertion sort, which produces the following canonical permutations:

Example 2.1.25. Consider the unique two-element list $[\bullet,\bullet] : \mathsf{List}(1)$, and denote $B_2 := \mathsf{Perm}([\bullet,\bullet],[\bullet,\bullet])$. The type B_2 is freely generated by



so it is akin to the braid group on two strands, since it remembers how often strands are braided over. Notably,



The unit type is trivially ordered, so $\mathsf{canonPerm} : B_2 \to B_2$ is defined. As implemented, isort is not stable and swaps equal elements: $\mathsf{isort}([\bullet,\bullet]) = [\bullet,\bullet]$. Therefore, $\mathsf{canonPerm}(\mathsf{s}) = \mathsf{s}^2 = \mathsf{swap}(\mathsf{swap}(\mathsf{id}))$, and $\mathsf{canonPerm}(\sigma) = \mathsf{s}^2$ for any $\sigma : B_2$. If it were stable, the canonical permutation would be $\mathsf{id} : \mathsf{Perm}([\bullet,\bullet],[\bullet,\bullet])$.

For details of the implementation and its computational behaviour, see module `Example` of [Ia, `Multiset.Ordering.Order`]. ⌟

2.2 Final M-coalgebras as a constructive taboo

In classical set theory, it is possible to show the existence of a final M-coalgebra via the *Adámek-Trnková fixpoint theorem* [AT90, Proposition IV.2.5.(iii)]. Our goal is to show that this approach is constructively taboo, in the following sense: In set theory, the carrier of the final M-coalgebra is a set L obtained as the limit of a ω-chain involving M. By the universal property of the limit, there exists a uniquely determined algebra map $\mathsf{in} : \mathsf{M}(\mathsf{L}) \to \mathsf{L}$. Classically, this map is an isomorphism, and its inverse is the structure map of the final coalgebra. We show that, in our constructive setting, in is an isomorphism of sets if and only if the non-constructive *lesser-limited principle of omniscience* holds. It asserts that for any sequence of boolean values $a : \mathbb{N} \to 2$, if a is true in at most one position, then either all even positions of a are false, or all odd positions of a are false:

Definition 2.2.1 (LLPO)**.** The *lesser-limited principle of omniscience* is the type

$$\mathsf{LLPO} :\equiv \prod_{a:\mathbb{N}\to 2} \mathsf{isProp}\left(\textstyle\sum_{n:\mathbb{N}} a_n = 1\right) \\ \to \left\| \left(\prod_{n:\mathbb{N}} \mathsf{isEven}(n) \to a_n = 0 \right) + \left(\prod_{n:\mathbb{N}} \mathsf{isOdd}(n) \to a_n = 0 \right) \right\|_{-1}$$

We say that LLPO holds if this type is inhabited. ⌟

This result is surprising. One would conjecture the finite multiset-functor to be a better behaved cousin of the finite powerset functor $\mathsf{P}_{\mathsf{fin}}$. The classical construction of the final $\mathsf{P}_{\mathsf{fin}}$-coalgebra as a $(\omega{+}\omega)$-limit [Wor05] depends on LLPO as well as the axiom of countable choice [Vel21, Theorem 10]. One would assume that the inability to construct this fixpoint is due to the "collapsing" nature of idempotency $x \cup x = x$ of union of finite sets. It turns out that commutativity $x \cup y = y \cup x$ lies at the heart of the impossibility, and that in the case of the $\mathsf{P}_{\mathsf{fin}}$-fixpoint, [Vel21, Theorem 10] holds even without assuming countable choice.

2.2.1 Final coalgebras from ω-limits

Let F be an endofunctor of sets. We recall the folklore set-theoretic construction of the final F-coalgebras from ω-limits. In a type-theoretic setting, this construction has been used, for example, by Altenkirch et al. to construct non-wellfounded trees from well-founded ones assuming *Axiom K* [Alt+15, § 7.2]. We restate it here to fix conventions and notation.

Definition 2.2.2. An ω-chain $X_\bullet \doteq (X, \pi)$ consists of a family of objects $X : \mathbb{N} \to \mathsf{hSet}$ and projections $\pi : \prod_{n:\mathbb{N}} X_{n+1} \to X_n$. As a type,

$$\omega\text{-}\mathsf{Chain} :\equiv \sum_{X:\mathbb{N}\to\mathsf{hSet}} \prod_{n:\mathbb{N}} X_{n+1} \to X_n$$ ⌟

Definition 2.2.3. Let $A : \mathsf{hSet}$ and $X_\bullet \doteq (X, \pi) : \omega\text{-}\mathsf{Chain}$. A *cone* over $X_\bullet$ with *apex* A is a family of maps $a : \prod_{n:\mathbb{N}} A \to X_n$ together with a family of paths $h : \prod_{n:\mathbb{N}} \pi_n \circ a_{n+1} = a_n$. We write

$$\mathsf{Cone}_{X_\bullet}(A) :\equiv \sum_{a:\prod_{n:\mathbb{N}} A\to X_n} \prod_{n:\mathbb{N}} \pi_n \circ a_{n+1} = a_n$$ ⌟

For any chain, we can define its *limit*.

Definition 2.2.4. The limit of the chain $X_\bullet \doteq (X, \pi)$ is the type

$$\mathsf{Lim}(X_\bullet) :\equiv \sum_{x:\prod_{n:\mathbb{N}} X_n} \prod_{n:\mathbb{N}} \pi_n(x_{n+1}) = x_n$$ ⌟

In proposition 2.2.7 we establish a universal property of this type, which justifies calling it a "limit".

The truncation level of $\mathsf{Lim}(X_\bullet)$ is bounded by that of X_n over all $n : \mathbb{N}$. In particular, since all X_n are sets, $\mathsf{Lim}(X_\bullet)$ is again a set. The type $\prod_n \pi_n(x_{n+1}) = x_n$ is a proposition, so $\mathsf{Lim}(X_\bullet)$ is a subtype of $\prod_{n:\mathbb{N}} X_n$, and we will write $x_n : X_n$ given some $x : \mathsf{Lim}(X_\bullet)$. This lets us conveniently state the following extensionality principle for limits:

Lemma 2.2.5. Let $X_\bullet \doteq (X, \pi) : \omega\text{-}\mathsf{Chain}$. For all $s, t : \mathsf{Lim}(X_\bullet)$, there is an equivalence of types $(s = t) \simeq (\forall n.\, s_n =_{X_n} t_n)$. □

Function extensionality implies that $\mathsf{Lim}(X_\bullet)$ sits at the apex of a cone over any chain $X_\bullet$:

Problem 2.2.6 (Universal cone)**.** For chain $X_\bullet \doteq (X, \pi)$, define a cone over $X_\bullet$ with apex $\mathsf{Lim}\, X_\bullet$, that is some

$$\ell_{X_\bullet} :\equiv (\ell, \ell^\pi) : \mathsf{Cone}_{X_\bullet}(\mathsf{Lim}\, X_\bullet)$$

Construction. The first projection out of $\mathsf{Lim}\, X_\bullet$ induces maps

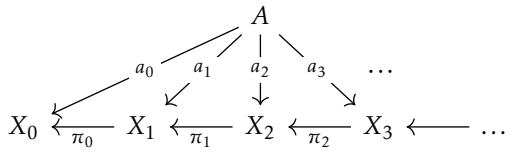
$$\ell : \textstyle\prod_{n:\mathbb{N}} \mathsf{Lim}\, X_\bullet \to X_n$$
$$\ell_n(x, \_) :\equiv x_n$$

These maps commute with the π_n by the second projection out of the limit, i.e. we have

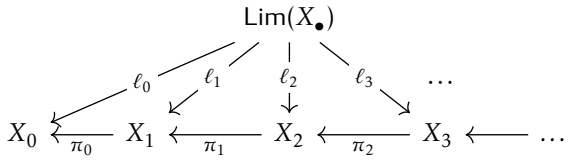
$$\ell^\pi : \textstyle\prod_{n:\mathbb{N}} \pi_n \circ \ell_{n+1} = \ell_n$$
$$\ell^\pi_n :\equiv \mathsf{funExt}\, \lambda(x, p).\, p_n$$

in which $p : \prod_n \pi_n(x_{n+1}) = x_n$. ⌟

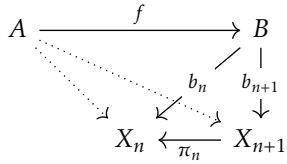
We draw a cone (A, a, h) over a chain (X, π) as diagram



The universal cone over (X, π) places $\mathsf{Lim}(X_\bullet)$ at the apex, connected to the chain by canonical projections $\ell_n : \mathsf{Lim}(X_\bullet) \to X_n$:



This cone is universal in the following sense: Given a map $f : A \to B$, we can pull back any cone $b : \mathsf{Cone}_{X_\bullet}(B)$ to a cone $f_* b : \mathsf{Cone}_{X_\bullet}(A)$ by precomposing with f:



Applied to the universal cone, this induces a map from functions $\mathsf{Lim}(X_\bullet)$ to cones. This assignment is universal and characterizes functions into the limit.

Proposition 2.2.7 (universal property)**.** For any chain $X_\bullet$ and $A : \mathsf{hSet}$, the map

$$\_{}_* \ell_{X_\bullet} : (A \to \mathsf{Lim}\, X_\bullet) \to \mathsf{Cone}_{X_\bullet}(A)$$

pulling back the universal cone $\ell_{X_\bullet}$ is an equivalence. □

We can now phrase the fixpoint theorem in type-theoretic terms:

Definition 2.2.8. Given an endofunctor F, define a chain $F^\bullet := (F^n(1), !^n)$ by iterating F on the unit type 1 and the unique map $! : F(1) \to 1$:

$$F^0 := 1 \qquad\qquad !^0 := !$$
$$F^{n+1} := F(F^n) \qquad\qquad !^{n+1} := F(!^n)$$ ⌟

It is the limit of this chain, $\nu F := \mathsf{Lim}(F^\bullet)$, that will be the carrier of the potential final F-coalgebra. We think of an element $t : \nu F$ as a possibly non-wellfounded tree, and of the projection $\ell_n(t) : F^n(1)$ as an approximation of t of depth at most n.

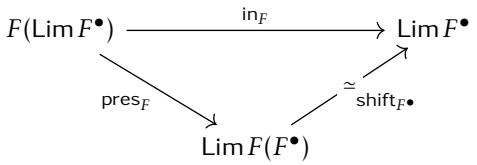
First, notice that F acts on chains $X_\bullet$: the *F-shifted chain* $F(X_\bullet) : \omega\text{-}\mathsf{Chain}$ is obtained by applying F at each step. Shifting a chain does not change its limit; there exists an equivalence of types $\mathsf{shift}_{X_\bullet} : \mathsf{Lim}\, X_\bullet \simeq \mathsf{Lim}\, F(X_\bullet)$. By the universal property of ω-chains, the limit of the shifted chain admits a map from $F(\mathsf{Lim}\, X_\bullet)$:

Problem 2.2.9. Define a map $\mathsf{pres}_F : F(\mathsf{Lim}\, X_\bullet) \to \mathsf{Lim}\, F(X_\bullet)$.

Construction. Each $F(\ell_n) : F(\mathsf{Lim}\, X_\bullet) \to F(X_n)$ is a projection in a cone with apex $F(\mathsf{Lim}\, X_\bullet)$, which is sufficient to define a map $F(\mathsf{Lim}\, X_\bullet) \to \mathsf{Lim}\, F(X_\bullet)$. ⌟

Taken together, pres_F and $\mathsf{shift}_{F^\bullet}$ equip νF with the structure of an *F-algebra*.

Definition 2.2.10. The F-algebra $\mathsf{in}_F : F(\nu F) \to \nu F$ is defined as the composite



⌟

In the following, we will say that F is *limit-preserving* if pres_F is an equivalence. Notably, this is a sufficient condition for a terminal F-coalgebra to exist:

Theorem 2.2.11 (Adámek-Trnková fixpoint theorem)**.** The algebra in_F is an equivalence if and only if F is limit-preserving. In this case $\mathsf{out}_F : \nu F \to F(\nu F)$ (the inverse of in_F) is the structure map of a final F-coalgebra.

Proof. The standard proof [AT90, Proposition IV.2.5.(iii)] is constructive. □

In univalent foundations, [ACS15, Theorem 7] establishes the above for coalgebras of *wild* polynomial endofunctors, from which this claim follows if restricted to h-sets. We discuss this result and its applications to wild categories in § 2.3.

2.2.2 Final M-coalgebras and LLPO

We are now ready to prove that the assumption of theorem 2.2.11 is a constructive taboo in case of the multiset functor: M is limit-preserving if and only if LLPO holds. First, observe that each set in the chain $1 \leftarrow M(1) \leftarrow M^2(1) \leftarrow \dots$ can be linearly ordered:

Lemma 2.2.12. For all $n : \mathbb{N}$, $\mathsf{M}^n(1)$ is linearly ordered. Consequently, the type $([k] \to \mathsf{M}^n(1))/\sim_k$ is a definable quotient.

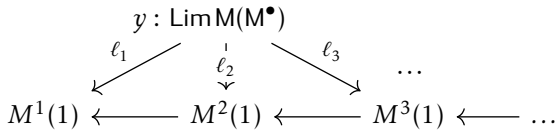
Proof. Define a linear order $<^n$ on M^n by induction: $<^0$ is the empty relation, and $<^{n+1} := <^n_{\mathsf{plex}} : \mathsf{M}(\mathsf{M}^n) \to \mathsf{M}(\mathsf{M}^n) \to \mathsf{Type}$ is a linear order on finite multisets (problem 2.1.22). □

This is enough to show that $\mathsf{pres}_{\mathsf{M}}$ is surjective in a strong sense:

Proposition 2.2.13 ([Ia, `Multiset.FMSet.Limit`])**.** The map $\mathsf{pres}_{\mathsf{M}}$ has a section

$$\mathsf{pres}^{-1} : \mathsf{Lim}\,\mathsf{M}(\mathsf{M}^\bullet) \to \mathsf{M}(\mathsf{Lim}\,\mathsf{M}^\bullet)$$

Proof. Let $y : \mathsf{Lim}\,\mathsf{M}(\mathsf{M}^\bullet)$; our goal is to construct a multiset $x : \mathsf{M}(\mathsf{Lim}\,\mathsf{M}^\bullet)$ such that $\mathsf{pres}(x) = y$. First, note that y lives in a cone over the (shifted) chain



hence all projections $\ell_n(y) : \mathsf{M}^n(1)$ must have the same cardinality $k : \mathbb{N}$ for $n \geq 1$, up to propositional equality. By an application of J, we can assume that for all $n \geq 1$ there exists some vector $v_n : ([k] \to \mathsf{M}^{n-1}(1))/{\sim_k}$ such that $\ell_n(y) \doteq (k, v_n)$. We define x in terms of its cardinality k and its members, that is

$$\begin{aligned} &x : \textstyle\sum_{k:\mathbb{N}}([k] \to \mathsf{Lim}\,\mathsf{M}^\bullet)\,/\sim_k \\ &x \coloneqq (k, [\bar{u}]) \end{aligned}$$

in which $\bar{u} : [k] \to \mathsf{Lim}\,\mathsf{M}^\bullet$ is a chosen representative of the equivalence class of members of x. From lemma 2.2.12 we know that the quotient is definable, and that there is a sequence of approximations

$$\begin{aligned} &w : \textstyle\prod_{n\geq 1}[k] \to \mathsf{M}^{n-1}(1) \\ &w_n \coloneqq \mathsf{rep}(v_n) \end{aligned}$$

Since y is a limit, it follows that $!^n(k, v_{n+1}) = (k, v_n)$ for all $n \geq 1$. This together with the fact that rep is a section of $[\_]$ induces an identification

$$[!^n \circ w_{n+1}] = [!^n \circ \mathsf{rep}(v_{n+1})] = v_n = [\mathsf{rep}(v_n)] = [w_n]$$

The quotient by $\sim_k$ is effective, hence $!^n \circ w_{n+1} \sim_k w_n$. We extract a canonical permutation by applying proposition 2.1.24: There is $\sigma : \mathbb{N} \to [k] \simeq [k]$ such that

$$\forall n.\, !^n \circ w_{n+1} = w_n \circ \sigma_n \tag{2.1}$$

To define $\bar{u} : [k] \to \mathsf{Lim}\,\mathsf{M}^\bullet$, it suffices to give, for each $i : [k]$, approximations $u_{i,n} : \mathsf{M}^n(1)$ such that $!^n \circ u_{i,n+1} = u_{i,n}$. First, fold over the permutation σ,

$$\begin{aligned} &\sigma^*_0 : [k] \simeq [k] && &&\sigma^*_{n+1} : [k] \simeq [k] \\ &\sigma^*_0 \coloneqq \mathsf{id} && &&\sigma^*_{n+1} \coloneqq \sigma_n^{-1} \circ \sigma^*_n \end{aligned}$$

and then permute w_n by σ^*_n:

$$\begin{aligned} &u_{i,0} : 1 && &&u_{i,n} : \mathsf{M}^n(1) \\ &u_{i,0} \coloneqq \bullet && &&u_{i,n} \coloneqq w_n(\sigma^*_n(i)) \end{aligned} \qquad \forall n \geq 1$$

We show that this sequence converges by applying equation (2.1):

$$!^n \circ u_{n+1} \doteq (!^n \circ w_{n+1} \circ \sigma_n^{-1}) \circ \sigma_n^* = w_n \circ \sigma_n^* \doteq u_n$$

It remains to show that this defines a section of pres, that is $\ell_n(\mathsf{pres}(x)) = \ell_n(y)$ for all $n : \mathbb{N}$. Unfolding definitions,

$$\begin{aligned} \ell_n(\mathsf{pres}(x)) &= \ell_n(\mathsf{pres}(k, [\bar{u}])) \\ &= (k, [\lambda i.\, u_{i,n}]) \\ &= (k, [w_n \circ \sigma_n^*]) \end{aligned} \qquad \text{and} \qquad \begin{aligned} \ell_n(y) &= (k, v_n) \\ &= (k, [\mathsf{rep}(v_n)]) \\ &= (k, [w_n]) \end{aligned}$$

we see that both sides coincide: $w_n \circ \sigma_n^*$ is a permutation of w_n, hence both are equal modulo $\sim_k$. □

By definition, the same must be the case for the canonical M-algebra in_M:

Corollary 2.2.14. The set $\nu\mathsf{M}$ is the carrier of an M-coalgebra

$$\mathsf{out}_\mathsf{M} : \nu\mathsf{M} \to \mathsf{M}(\nu\mathsf{M})$$

This coalgebra is a section of in_M, that is $\mathsf{in}_\mathsf{M}(\mathsf{out}_\mathsf{M}(t)) = t$ for all $t : \nu\mathsf{M}$. □

It is injectivity of pres_M that prevents out_M from being a (necessarily largest) M-fixpoint:

Theorem 2.2.15. The following are equivalent propositions:

1. $\mathsf{in}_\mathsf{M} : \mathsf{M}(\nu\mathsf{M}) \to \nu\mathsf{M}$ is an isomorphism (and out_M is the final M-coalgebra).
2. $\mathsf{pres}_\mathsf{M} : \mathsf{M}(\mathsf{Lim}\,\mathsf{M}^\bullet) \to \mathsf{Lim}\,\mathsf{M}(\mathsf{M}^\bullet)$ is injective.
3. LLPO holds.

Proof. Since pres_M admits a section the first two points are equivalent by corollary 2.2.14. We establish equivalence with LLPO in the remainder of this chapter, namely lemma 2.2.17, propositions 2.2.18 and 2.2.19, and corollary 2.2.21, as depicted in figure 2.1. □

The proof of theorem 2.2.15 factors through notions of "completeness" of subsets in the limit of $\mathsf{M}^\bullet$. Completeness is a property of any ω-chain:

Definition 2.2.16. For a chain $X_\bullet = (X, \pi) : \omega\text{-}\mathsf{Chain}$, the following are properties of its limit:

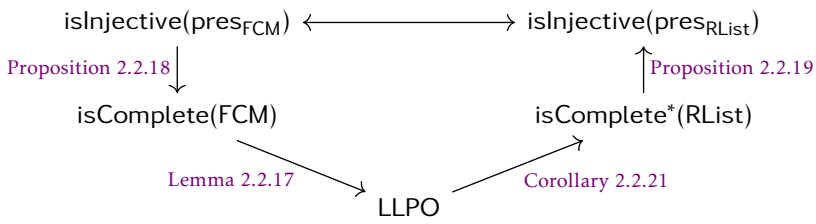


Figure 2.1: Proof strategy to show that LLPO is equivalent to injectivity of $\mathsf{pres}_{\mathsf{M}}$. Injectivity of $\mathsf{pres}_{\mathsf{M}}$ is independent of the choice of representative, FCM or RList.

1. *Binary subsets are complete:*

$$\mathsf{isComplete}(X_\bullet) :\equiv \prod_{x,u,v:\mathsf{Lim}\,X} (\forall n.\, \pi_n(x) = \pi_n(u) + \pi_n(x) = \pi_n(v)) \to \|x = u + x = v\|_{-1}$$

2. *Finite subsets are complete:*

$$\mathsf{isComplete}^*(X_\bullet) :\equiv \prod_{x:\mathsf{Lim}\,X} \prod_{ys:\mathsf{List}(\mathsf{Lim}\,X)} (\forall n.\, \|\pi_n(x) \in \mathsf{map}_{\pi_n}(ys)\|_{-1}) \to \|x \in ys\|_{-1}$$

⌟

If we think of the projections $\pi_n : \mathsf{Lim}\,X \to X_n$ as approximations, then completeness of binary subsets says that, if $x : \mathsf{Lim}\,X$ is approximated by either u or v at each step n, then x itself must be merely equal to one of the two. This closely resembles the notion of completeness of subspaces of a metric space X, where $\{u, v\} \subset X$ is complete if every Cauchy sequence taking values only in u or v converges to either u or v. A result of constructive analysis asserts that binary subsets of $\mathbb{R}$ are complete if and only if LLPO holds [Man88, Theorem 4.1].[1] We establish a similar result for $\mathsf{Lim}(\mathsf{M}^\bullet)$ by adapting [Vel21, Theorem 7].

Lemma 2.2.17. If binary subsets of νFCM are complete, then LLPO holds.

Proof. Given a sequence $a : \mathbb{N} \to 2$, the idea is to construct two trees. The first, $\mathsf{long} : \nu\mathsf{FCM}$ is the infinitely deep unary tree, specified coinductively by $\mathsf{long} = \eta(\mathsf{long})$. The second, $\mathsf{long?}(a) : \nu\mathsf{FCM}$, consists of nested applications of η as well, but the nesting stops with ε after n steps if $a_n = 1$. From this, we build a third tree $x : \nu\mathsf{FCM}$ as follows: For all n, the approximation x_n is long_n, unless

[1] We are curious whether the analogy can be made precise by exhibiting $\mathbb{R}$ as a final coalgebra of some endofunctor, perhaps by Pavlović and Pratt's construction [PP99] of the continuum as a final coalgebra of $X \mapsto X \cdot \omega$.

there is some *even* $k \leq n$ such that $a_n = 1$, then it is $\mathsf{long?}_n$. By completeness it follows that merely either $x = \mathsf{long}$ or $x = \mathsf{long?}(a)$. In the first case we can deduce $a_n = 0$ for all even n — if this were not the case we could prove $\mathsf{long} = \mathsf{long?}$, which is contradictory since in this case, long? has finite depth. In the second case show that $a_n = 0$ for all odd n by a similar argument. The detailed proof is available at [Ia, `Multiset.FCM.Limit`]. □

Representing finite multisets as free commutative monoids, we show that injectivity of $\mathsf{pres}_{\mathsf{FCM}}$ implies that binary subsets are complete, and that LLPO follows.

Proposition 2.2.18. If $\mathsf{pres}_{\mathsf{FCM}}$ is injective, then binary subsets of $\nu\mathsf{FCM}$ are complete.

Proof. Let $x, u, v : \nu\mathsf{FCM}$. Write x_n for $\ell_n(x)$, and similarly for u and v. Denote by $\wr x, y \int := \eta(x) \oplus \eta(y)$ a binary multiset, and element-hood by $z \in \wr x, y \int := \|(z = x) + (z = y)\|_{-1}$. We are going to construct some $\bar{x} : \nu\mathsf{FCM}$ such that

$$x \in \wr x, \bar{x} \int \qquad \text{and} \qquad \wr x, \bar{x} \int = \wr u, v \int,$$

hence $\|(x = u) + (x = v)\|_{-1}$. We define $\bar{x}$ in terms of its projections $\bar{x}_n : \mathsf{FCM}^n(1)$, by inspecting the assumption $\mathsf{approx} : \forall n.\, x_n = u_n + x_n = v_n$:

$$\bar{x}_n := \begin{cases} v_n & \text{if } \mathsf{approx}_n \doteq \mathsf{inl}(\_ : x_n = u_n) \\ u_n & \text{if } \mathsf{approx}_n \doteq \mathsf{inr}(\_ : x_n = v_n) \end{cases}$$

By examining approx_n and approx_{n+1} it is straightforward to show that $\bar{x}_n$ satisfies the limit property $!^n(\bar{x}_{n+1}) = \bar{x}_n$ for all $n : \mathbb{N}$. Obviously, $x \in \wr x, \bar{x} \int$.

Now we apply injectivity of of $\mathsf{pres}_{\mathsf{FCM}}$: in order to show $\wr x, \bar{x} \int = \wr u, v \int$, it suffices to give a path

$$p : \mathsf{pres}(\wr x, \bar{x} \int) = \mathsf{pres}(\wr u, v \int)$$

This is a path in a limit, so by extensionality (lemma 2.2.5) it is enough to show

$$\forall n : \mathbb{N}.\, \ell_n(\mathsf{pres} \wr x, \bar{x} \int) = \ell_n(\mathsf{pres} \wr u, v \int)$$

By definition of pres, this type reduces to

$$\forall n : \mathbb{N}.\, \wr x_n, \bar{x}_n \int = \wr u_n, v_n \int$$

For any $n : \mathbb{N}$, inspect approx_n. Either $\mathsf{approx}_n \doteq \mathsf{inl}(p : x_n = u_n)$ and hence

$$\wr x_n, \bar{x}_n \int = \wr x_n, v_n \int \overset{p}{=} \wr u_n, v_n \int$$

or $\mathsf{approx}_n \doteq \mathsf{inr}(q : x_n = v_n)$, and in that case

$$\wr x_n, \bar{x}_n \wr = \wr x_n, u_n \wr \overset{q}{=} \wr v_n, u_n \wr = \wr u_n, v_n \wr$$

by commutativity of $\oplus$. In either case this completes the proof showing that x is merely either u or v. □

In the other direction, we represent multisets by lists up to permutation to derive injectivity of pres.

Proposition 2.2.19. If finite subsets of $\nu\mathsf{RList}$ are complete, then $\mathsf{pres}_{\mathsf{RList}}$ is injective.

Proof. Assuming completeness of finite subsets, our goal is to show that

$$\prod_{xs,ys:\mathsf{RList}(\nu\mathsf{RList})} \mathsf{pres}(xs) = \mathsf{pres}(ys) \to xs = ys$$

The conclusion is a proposition, so it suffices to establish this on representatives:

$$\prod_{xs,ys:\mathsf{List}(\nu\mathsf{RList})} \mathsf{pres}[xs] = \mathsf{pres}[ys] \to \mathsf{Relator}_=(xs, ys)$$

For all $xs, ys : \mathsf{List}(\nu\mathsf{RList})$ we factor the proof into two steps, namely

$$\begin{array}{ccc} \mathsf{pres}[xs] = \mathsf{pres}[ys] & \dashrightarrow & \mathsf{Relator}_=(xs, ys) \\ & \searrow \quad \nearrow & \\ & \forall n.\, \mathsf{Relator}_=(\ell_n^*(xs), \ell_n^*(ys)) & \end{array}$$

where $\ell_n^* \coloneqq \mathsf{map}_{\ell_n} : \mathsf{List}(\nu\mathsf{RList}) \to \mathsf{List}(\mathsf{M}^n(1))$. The first step follows since $\mathsf{Relator}_=$ is an effective quotient; it is enough to show that $[\ell_n^*(xs)] = [\ell_n^*(ys)]$ implies $\mathsf{Relator}_=(\ell_n^*(xs), \ell_n^*(ys))$. To prove the second implication, it suffices to show that

$$\Big(\forall n.\, \mathsf{DRelator}_=(\ell_n^*(xs), \ell_n^*(ys))\Big) \to \mathsf{DRelator}_=(xs, ys)$$

since $\mathsf{Relator}_=$ is the symmetrization of $\mathsf{DRelator}_=$. We show this by induction on xs. The empty case is absurd. In the ::-case we show that

$$\forall n. \|\ell_n(x) \in \ell_n^*(ys)\|_{-1}$$

Now our completeness-assumption applies and we can conclude

$$\|x \in ys\|_{-1},$$

hence $\mathsf{DRelator}_=(x :: xs, ys)$. □

The missing piece to complete theorem 2.2.15 holds in surprising generality. We notice this by introducing a cut in the proof of [I, Theorem 9]: Instead of proving LLPO $\to$ isInjective($\mathsf{pres}_{\mathsf{RList}}$) directly, we observe that the proof already factors through isComplete*(RList), and that the implication from this to LLPO only relies on $\mathsf{RList}^n(1)$ being discrete for all $n : \mathbb{N}$.

Theorem 2.2.20. Let $X_\bullet \doteq (X, \pi) : \omega\text{-Chain}$ such that X_n is discrete for all n. Then LLPO implies isComplete$^*(X_\bullet)$.

Proof. Let us sketch the proof. Assume LLPO; let $x : \mathsf{Lim}\, X_\bullet$, $ys : \mathsf{List}(\mathsf{Lim}\, X_\bullet)$, and $h : \forall n. \|\pi_n(x) \in \mathsf{map}_{\pi_n}(ys)\|_{-1}$. If ys is the empty list, then $h(0) : \|\pi_0(x) \in [\,]\|_{-1}$ is absurd, so the claim follows. If $ys \doteq y :: ys'$, we construct a sequence $a : \mathbb{N} \to 2$ such that for all $n : \mathbb{N}$,

$$\text{is even } n \implies a_n = \begin{cases} 1 & \text{if } \pi_n(x) \neq \pi_n(y) \\ 0 & \text{otherwise} \end{cases}$$

$$\text{is odd } n \implies a_n = \begin{cases} 1 & \text{if } \pi_n(x) \in \mathsf{map}_{\pi_n}(ys) \\ 0 & \text{otherwise} \end{cases}$$

From this we derive a second sequence $a' : \mathbb{N} \to 2$ that satisfies the following:

- $a'_n = 0$ for even n,
- $a'_n = a_n$, for odd n,
- except when there is some odd $k < n$ such that $a_k = 1$, then $a'_n = 0$.

Evidently, a' is 1 for at most one position — if this happens, it happens at smallest odd k for which $a_k = 1$. LLPO then tells us merely whether it is the even or the odd positions of a' that are 0. By analyzing all possible cases (using the assumption h), we conclude that either $\|x = y\|_{-1}$ or $\|x \in ys'\|_{-1}$. This is enough to show that $\|x \in y :: ys'\|_{-1}$. □

Corollary 2.2.21. If LLPO holds, then isComplete$^*(\mathsf{RList}^\bullet)$.

Proof. By lemma 2.2.12, $\mathsf{RList}^n(1)$ is linearly ordered for all n, hence discrete, and theorem 2.2.20 applies. □

Remark 2.2.22. Theorem 2.2.15 shows that, constructively, the fixpoint construction over ω-chains is not strong enough to produce a final M-coalgebra. Inspecting the proofs, we observe that M is not even the minimal example of a functor exhibiting this behaviour. Consider the functor $\mathsf{U}(X) \coloneqq 1 + X^2/\sim$ generated by $(x, y) \sim (y, x)$. Its limit is a type of unordered binary trees with unlabeled

leaves. This functor has enough structure to define both the "long trees" of lemma 2.2.17 and the binary multisets of proposition 2.2.18, thus injectivity of $\mathsf{pres}_{\mathsf{U}}$ implies LLPO. In the other direction theorem 2.2.20 still applies, and so does the proof of proposition 2.2.19 if we think of U as RList restricted to lists of length 0 or 2. ⌟

2.2.3 As a quotient of ordered trees

Instead of considering a type of unordered trees, quotiented at each level, we investigate whether it is possible to define the final M-coalgebra by quotienting a type of non-wellfounded *ordered* trees by a suitable relation. We know that, constructively, the list functor admits a final coalgebra: List arises as a polynomial endofunctor of sets, and a standard argument[2] shows that $\mathsf{pres}_{\mathsf{List}}$ preserves limits, hence theorem 2.2.11 applies.

By choosing a suitable relation R on $\nu\mathsf{List} := \mathsf{Lim}\,\mathsf{List}^\bullet$, we obtain a type of *unordered* trees $\nu\mathsf{List}/R$. The obvious candidate for a relation defining unordered trees is that of *bisimilarity*: Two trees in $\nu\mathsf{List}$ are *bisimilar* if at all depths, they are equal up to a level-wise permutation. We then attempt to lift $\mathsf{out}_{\mathsf{List}} : \nu\mathsf{List} \to \mathsf{List}(\nu\mathsf{List})$ to an M-coalgebra $\mathsf{fix}^+ : \nu\mathsf{List}/R \to \mathsf{M}(\nu\mathsf{List}/R)$. Unfortunately, this fails in the same way that $\mathsf{pres}_{\mathsf{M}}$ fails to be injective: In order for fix^+ to be well-defined, $\mathsf{out}_{\mathsf{List}}$ needs to preserve R, but this is the case if and only if LLPO holds.

We define bisimilarity as the limit of a chain indexed by two non-wellfounded trees in $\nu\mathsf{List}$. First, we express the relation witnessing that two trees of a fixed maximum depth are level-wise permutations of each other:

Definition 2.2.23. For all $n : \mathbb{N}$, the relation $\approx_n$ on $\mathsf{List}^n(1)$ is defined inductively:

$$\begin{array}{ll} \_ \approx_0 \_ : 1 \to 1 \to \mathsf{Prop} & \_ \approx_{n+1} \_ : \mathsf{List}(\mathsf{List}^n(1)) \to \mathsf{List}(\mathsf{List}^n(1)) \to \mathsf{Prop} \\ s \approx_0 t := 1 & s \approx_{n+1} t := \mathsf{Relator}_{\approx_n}(s,t) \end{array}$$ ⌟

For any pair of trees, this defines a chain:

Problem 2.2.24. Let $s, t : \nu\mathsf{List}$. Define a chain of propositions

$$s \approx_\bullet t := \quad \ell_0(s) \approx_0 \ell_0(t) \xleftarrow{\beta_0} \ell_1(s) \approx_1 \ell_1(t) \xleftarrow{\beta_1} \ell_2(s) \approx_2 \ell_2(t) \xleftarrow{\beta_2} \dots$$

Construction. We have to ensure that $\beta_n : \ell_{n+1}(s) \approx_{n+1} \ell_{n+1}(t) \to \ell_n(s) \approx_n \ell_n(t)$. Trivially, β_0 holds. The successor-case follows from basic properties of Relator, and the fact that s (respectively t) lives in a limit, i.e. satisfies $!^n(\ell_{n+1}(s)) = \ell_n(s)$. ⌟

[2] For example, [Alt+15, Proposition 7.2]. Or theorem 2.3.1, as we will see later.

The limit of this indexed chain defines a relation on $\nu\mathsf{List}$, expressing the intuition that two non-wellfounded trees are bisimilar if at each level of approximation they are level-wise permutations of each other.

Definition 2.2.25. Bisimilarity of ordered trees is the type of limits

$$\begin{aligned}&\mathsf{Bisim} : \nu\mathsf{List} \to \nu\mathsf{List} \to \mathsf{Type}\\ &\mathsf{Bisim}(s,t) :\equiv \mathsf{Lim}(s \approx_\bullet t)\end{aligned}$$ ⌟

To see whether $\mathsf{out}_{\mathsf{List}}$ lifts to a coalgebra structure on $\nu\mathsf{List}/\mathsf{Bisim}$, we first investigate whether it defines a coalgebra of setoids:

Definition 2.2.26. The category Setoid has as objects pairs $(X, \sim_X)$ of a set X and a propositionally-valued equivalence relation $\sim_X$ on X. Morphisms are relation-preserving maps,

$$\mathsf{Setoid}_1((X,\sim_X),(Y,\sim_Y)) :\equiv \sum_{f:X\to Y} \prod_{x_0,x_1:X} x_0 \sim_X x_1 \to f(x_0) \sim_Y f(x_1)$$ ⌟

Problem 2.2.27. Extend List to a functor $\mathsf{List}_{\mathsf{rel}} : \mathsf{Setoid} \to \mathsf{Setoid}$ with its action on objects given by

$$\mathsf{List}(X, \sim_X) :\equiv (\mathsf{List}(X), \mathsf{Relator}_{\sim_X})$$

In particular, $\mathsf{out}_{\mathsf{List}} : \nu\mathsf{List} \to \mathsf{List}(\nu\mathsf{List}))$ is a $\mathsf{List}_{\mathsf{rel}}$-coalgebra if and only if it is a morphism of setoids. This, however, is plagued by the same issues as trying to prove that $\mathsf{pres}_{\mathsf{M}}$ is injective:

Theorem 2.2.28. The following are equivalent propositions:

1. $\mathsf{out}_{\mathsf{List}}$ is the final $\mathsf{List}_{\mathsf{rel}}$-coalgebra.
2. $\mathsf{out}_{\mathsf{List}}$ is a morphism of setoids, i.e.
$$\prod_{s,t:\nu\mathsf{List}} \mathsf{Bisim}(s,t) \to \mathsf{Relator}_{\mathsf{Bisim}}(\mathsf{out}_{\mathsf{List}}(s), \mathsf{out}_{\mathsf{List}}(t))$$
3. LLPO holds.

Proof. The proof is similar to that of theorem 2.2.15, and factors through properties of the limit-preservation map $\mathsf{pres}_{\mathsf{List}}$. To show that LLPO holds, we use the assumption that $\mathsf{out}_{\mathsf{List}}$ preserves bisimilarity to conclude that certain lists of two elements are related by $\mathsf{Relator}_{\mathsf{Bisim}}$, just like in lemma 2.2.17. If we assume LLPO, we can adapt the proof of proposition 2.2.19: there, LLPO is used to ensure that a map between set-quotients is well-defined, i.e. preserves a certain relation. □

Theorem 2.2.29. If any of the conditions of theorem 2.2.28 hold, then $\mathsf{out}_{\mathsf{List}}$ lifts to an M-fixpoint $\mathsf{fix} : \nu\mathsf{List}\,/\,\mathsf{Bisim} \simeq \mathsf{M}(\nu\mathsf{List}\,/\,\mathsf{Bisim})$.

Proof. On one hand, theorem 2.2.28.(2) implies that $\mathsf{out}_{\mathsf{List}}$ lifts to a well-defined map of quotients

$$\mathsf{fix}^{+} : \nu\mathsf{List}\,/\,\mathsf{Bisim} \to \mathsf{RList}(\nu\mathsf{List}\,/\,\mathsf{Bisim})$$

On the other, it follows from theorem 2.2.28.(1), that $\mathsf{out}_{\mathsf{List}}$ has (by Lambek's lemma) an inverse $\mathsf{in}_{\mathsf{List}}$ as a morphism of setoids. Since $\mathsf{List}(X)\,/\,\mathsf{Relator}_R$ is an effective quotient for *any* setoid (X, R), this again lifts to a map

$$\mathsf{fix}^{-} : \mathsf{RList}(\nu\mathsf{List}\,/\,\mathsf{Bisim}) \to \nu\mathsf{List}\,/\,\mathsf{Bisim}$$

Both maps arise from mutually inverse morphisms of setoids, hence they must be inverses of each other up to propositional equality, i.e. induce an equivalence. □

It is not obvious that fix is the largest M-fixpoint, hence the final M-coalgebra. Without further assumptions, it seems impossible to define mediating morphisms from other coalgebras into fix, let alone prove that those are unique. Just like in the case of the finite powerset functor [Vel17, Theorem 4], however, this becomes possible if we assume that the axiom of choice for h-sets (definition 1.4.16) holds in the following, equivalent form:

Definition 2.2.30. Let $A, B : \mathsf{hSet}$ and R a relation on B. The pointwise lifting of R to $A \to B$ is the relation

$$\begin{aligned} R^* &: (A \to B) \to (A \to B) \to \mathsf{Prop} \\ R^*(f, g) &:\equiv \textstyle\prod_{a:A} R(f(a), g(a)) \end{aligned}$$

⌟

Problem 2.2.31. Given $A, B : \mathsf{hSet}$ and a relation R on B, define

$$\mathsf{ev}_R : (A \to B)\,/\,R^* \to A \to (B\,/\,R)$$

Construction. By induction on the quotient, define $\mathsf{ev}_R[f] :\equiv \lambda a.\,[f(a)]$. ⌟

Lemma 2.2.32. Let $A, B : \mathsf{hSet}$ and R a relation on B.

1. If R is effective, then ev_R is an embedding.
2. Assuming the axiom of choice for h-sets holds, ev_R is an equivalence.

Proof. Since $B\,/\,R$ is a set, ev_R is an embedding if it is injective, i.e.

$$\prod_{x,x':(A\to B)/R^*} \mathsf{ev}_R(x) = \mathsf{ev}_R(x') \to x = x'$$

By induction on x and x', we can assume $\mathsf{ev}_R[f] = \mathsf{ev}_R[f']$ for some $f, f' : A \to B$. There is a chain of equivalences

$$\mathsf{ev}_R[f] = \mathsf{ev}_R[f'] \simeq \prod_a [fa] = [f'a] \overset{(*)}{\simeq} \prod_a R(fa, f'a) \simeq R^*(f, f')$$

where $(*)$ is effectiveness of R. Therefore, $[f] =_{(A\to B)/R^*} [f']$ as desired. That ev_R is also surjective is equivalent to the axiom of choice by [Vel21, Theorem 2.1]. □

Theorem 2.2.33. Assuming the axiom of choice for sets, fix of theorem 2.2.29 is the structure map of the final M-coalgebra.

Proof. We use the presentation of M by RList. Abbreviate $U := \nu\mathsf{List}\,/\,\mathsf{Bisim}$ and $R := \mathsf{Relator}_=$. Given any coalgebra $c : C \to \mathsf{RList}(C)$, our goal is to show that the type of coalgebra morphisms from (C, c) to $(\nu\mathsf{List}\,/\,\mathsf{Bisim}, \mathsf{fix})$ is contractible. We define a coalgebra map $u(c) : C \to U$ by factoring as follows:

$$\begin{array}{ccc} (C \to \mathsf{List}(C)\,/\,R) & \overset{u}{\dashrightarrow} & (C \to U) \\ & {\scriptstyle \mathsf{ev}_R^{-1}} \swarrow \; \simeq \; \nearrow {\scriptstyle \mathsf{ev}_R} \quad \searrow {\scriptstyle u'} & \\ & (C \to \mathsf{List}(C))\,/\,R^* & \end{array}$$

The equivalence on the left is obtained from the axiom of choice applied to lemma 2.2.32, and u' by induction and the corecursor for $\nu\mathsf{List}$,

$$\begin{aligned} u' &: (C \to \mathsf{List}(C))\,/\,R^* \to (C \to \nu\mathsf{List}\,/\,R) \\ u'[f] &:= \lambda x.\,[\mathsf{corec}_{\mathsf{List}}(f, x)] \end{aligned}$$

since each $f : C \to \mathsf{List}(C)$ is a List-coalgebra. This is well-defined; it is straightforward to show that, if $R^*(f, f')$, then $\mathsf{Bisim}(\mathsf{corec}_{\mathsf{List}}(f, x), \mathsf{corec}_{\mathsf{List}}(f', x))$ for all $x : C$. It remains to show that $u(c)$ is the unique such coalgebra morphism, but this can be lifted from the universal property of $\mathsf{corec}_{\mathsf{List}}$. □

2.3 Constructing unordered trees in groupoids

The results of the previous section are an indication that set-based definitions of finite, unordered collections are not fit to describe possibly non-wellfounded, unordered trees in a constructive setting. In this section we study a groupoid-based definition and — following ideas of Kock [Koc12], Finster et al. [Fin+21], and Gylterud [Gyl11] — argue that a better perspective on finite, unordered collections in a univalent setting is to define them as groupoids instead. The

rationale is that identifications of multisets are permutations, and that these should inherently be treated as *data*. Instead of viewing them as quotients of lists, thereby forgetting about this data, we ought to define a type whose identifications are equivalent to a *set* of permutations. Any constructions based on such a type are necessarily homotopy-coherent, hence they automatically respect this extra data, making them invariant under permutation for free.

In this section, we define unordered collections as families over finite sets. To distinguish this proof-relevant notion from multisets, we call such collections *bags*. Unfortunately, quantifying over the universe of finite sets results in a type former taking values in a successor universe. We give a second equivalent, but small presentation, based on a small axiomatization of the universe of finite sets [Fin+21, §5.2].

The key insight of this section is that both types of bags are represented by $(1,0)$-truncated containers. As such, both admit a final coalgebra — not in the 1-category of h-sets, but in the wild category of types. The definitions and notation of § 2.2.1 make sense for arbitrary ω-chains in Type. In this setting, Ahrens et al. prove that for any container $(S \lhd P)$, the ω-limit of the chain $[\![S \lhd P]\!]^\bullet$ with respect to the wild extent-functor $[\![S \lhd P]\!] : \mathsf{Type} \to \mathsf{Type}$ is the carrier of the final coalgebra — independently of the container's truncation level:

Theorem 2.3.1 ([ACS15, Theorem 7])**.** For all $(S \lhd P) : \mathsf{Cont}$, the type $\nu[\![S \lhd P]\!] \doteq \mathsf{Lim}([\![S \lhd P]\!]^\bullet)$ is the carrier of the final $[\![S \lhd P]\!]$-coalgebra. Its structure map $\mathsf{out}_{[\![S \lhd P]\!]}$ is the inverse of $\mathsf{in}_{[\![S \lhd P]\!]}$ as in definition 2.2.10. □

As noted in § 1.5, wild functors do not always give rise to a wild category of coalgebras. Container functors $[\![S \lhd P]\!] : \mathsf{Type} \to \mathsf{Type}$ are however well-behaved, and do admit a wild category of coalgebras, as a direct argument shows.

2.3.1 Finite bags, large

Recall that a finite set is a type merely equivalent to some standard finite type (cf. definition 1.4.10). A finite collection of elements of a type X is a pair consisting of a finite indexing set B, and a function $B \to X$ picking the elements.

Definition 2.3.2. The large type of *totes* (or *large bags*) over $X : \mathsf{Type}$ is

$$\mathsf{Tote}(X) : \mathsf{Type}^+$$
$$\mathsf{Tote}(X) :\equiv \sum_{B:\mathsf{FinSet}} (B \to X)$$ ⌟

Univalence implies that the path type $(B, v) = (C, w)$ in $\mathsf{Tote}(X)$ is equivalent to the type of dependent pairs

$$\sum_{\sigma : B \simeq C} v = w \circ \sigma \tag{2.2}$$

Equivalences $B \simeq C$ form a set, indicating that $\mathsf{Tote}(X)$ itself is not a set. In general, it is at least a groupoid.

Proposition 2.3.3. If X is a groupoid, then $\mathsf{Tote}(X)$ is a groupoid as well. □

For standard finite types, the propositional truncation of the type in equation (2.2) is exactly equality of multisets (cf. definition 2.1.11), so we expect an equivalence $\mathsf{FMSet}(X) \simeq \|\mathsf{Tote}(X)\|_0$. To show that this is indeed the case, we have to make sure to *coherently* apply the induction principles of set-quotients and propositional truncation:

Theorem 2.3.4. For any $X : \mathsf{Type}$, there is an equivalence $\mathsf{FMSet}(X) \simeq \|\mathsf{Tote}(X)\|_0$.

Proof. We construct an explicit isomorphism $\mathsf{FMSet}(X) \cong \|\mathsf{Tote}(X)\|_0$. The forward direction is given by induction on $\mathsf{FMSet}(X)$: Let

$$f : \textstyle\prod_{n:\mathbb{N}} ([n] \to X) \to \|\mathsf{Tote}(X)\|_0$$
$$f_n(v) :\equiv |([n], v)|_0$$

If we can show that f respects $(\sim)$, i.e. $\prod_{n:\mathbb{N}} \prod_{v,w:[n]\to X} v \sim_n w \to f_n(v) = f_n(w)$, then f extends to a well-defined function $\mathsf{toTote} : \mathsf{FMSet}(X) \to \|\mathsf{Tote}(X)\|_0$. Let $v, w : [n] \to X$. Since $f_n(v) = f_n(w)$ is a proposition, we can assume access to a permutation σ for which $r : v = w \circ \sigma$. Univalence yields a path $\mathsf{ua}(\sigma) : [n] = [n]$, and transporting r along it gives us an identification $([n], v) =_{\mathsf{Tote}\,X} ([n], w)$, hence $f_n(v) = |([n], v)|_0 = |([n], w)|_0 = f_n(w)$, as desired.

To define an inverse $\mathsf{toFMSet} : \|\mathsf{Tote}(X)\|_0 \to \mathsf{FMSet}(X)$, notice that $\mathsf{FMSet}(X)$ is a set; it suffices to provide $\mathsf{toFMSet}^* : \mathsf{Tote}(X) \to \mathsf{FMSet}(X)$. Let $B : \mathsf{FinSet}$ with cardinality n and evidence $e : \|B \simeq [n]\|_{-1}$. Given elements $v : B \to X$, we would like to define

$$\mathsf{toFMSet}^*(B, v) :\equiv (n, ?)$$

by induction on $e : \|B \simeq [n]\|_{-1}$. But $? : ([n] \to X)/\sim_n$ lives in a set, so we cannot apply the standard induction principle for propositional truncation directly. We can however apply a derived recursion principle [CKV15, Corollary 2]. In the diagram

$$\begin{array}{ccc} \|B \simeq [n]\|_{-1} & \overset{g}{\dashrightarrow} & ([n] \to X)/\sim_n \\ \lfloor - \rfloor_{-1} \uparrow & \nearrow_{g^*} & \\ B \simeq [n] & & \end{array} \tag{2.3}$$

the dashed map g exists if g^* is weakly constant, that is $\prod_{\alpha,\beta} g^*(\alpha) = g^*(\beta)$. Let $g^*(\alpha) :\equiv [v \circ \alpha]$. This is obviously weakly constant, since for any $\alpha, \beta : [n] \simeq [n]$,

$$v \circ \alpha = v \circ \beta \circ (\beta^{-1} \circ \alpha),$$

hence $v \circ \alpha \sim_n v \circ \beta$, and thus $[v \circ \alpha] = [v \circ \beta]$. We complete the definition of $\mathsf{toFMSet}^*$ by setting $? := g(e)$.

Proving that $\mathsf{toFMSet} \circ \mathsf{toTote} = \mathsf{id}_{\mathsf{FMSet}(X)}$ is straightforward. Proving that $\mathsf{toTote} \circ \mathsf{toFMSet} = \mathsf{id}_{\|\mathsf{Tote}(X)\|_0}$ reduces to showing that $v \circ \alpha \sim_n v$ for any $v : B \to X$ and $\alpha : [n] \simeq B$, which is also direct. □

2.3.2 Finite bags, small

As noted in the introduction, $X : \mathsf{Type}$ and $\mathsf{Tote}(X) : \mathsf{Type}^+$ live in different universes. Since it is our goal to define the final coalgebra of finite bags as a ω-limit, we would have to define a chain

$$1 \leftarrow \mathsf{Tote}(X) \leftarrow \mathsf{Tote}^2(X) \leftarrow \mathsf{Tote}^3(X) \leftarrow \dots$$

We cannot, however, expect the iteration $\lambda n.\, \mathsf{Tote}^n(1)$ to live in a finite successor universe of Type. In principle, we could define $\mathsf{Tote}^+ : \mathsf{Type}^+ \to \mathsf{Type}^+$, which would allow defining the iteration starting from a large unit type $1^+ : \mathsf{Type}^+$, resulting in a large limit $\mathsf{Lim}((\mathsf{Tote}^+)^\bullet) : \mathsf{Type}^+$. Instead, we follow [Fin+21, §5.2] and define a small presentation of FinSet — a higher inductive type Bij : Type equivalent to FinSet.

Definition 2.3.5 ([Fin+21, Definition 23])**.** The higher inductive type Bij : Type has the following constructors:

- A point constructor:
$$\frac{n : \mathbb{N}}{\mathsf{obj}(n) : \mathsf{Bij}}$$
- A path constructor:
$$\frac{m, n : \mathbb{N} \quad \alpha : [m] \simeq [n]}{\mathsf{hom}(\alpha) : \mathsf{obj}(m) = \mathsf{obj}(n)}$$
- Higher path constructors:
$$\frac{m, n, o : \mathbb{N} \quad \alpha : [m] \simeq [n] \quad \beta : [n] \simeq [o]}{\mathsf{seq} : \mathsf{hom}(\alpha; \beta) = \mathsf{hom}(\alpha) \cdot \mathsf{hom}(\beta)} \qquad \frac{}{\mathsf{trunc} : \mathsf{isGroupoid}(\mathsf{Bij})}$$

⌟

Proposition 2.3.6 ([Fin+21, Theorem 15])**.** There is an equivalence $\mathsf{Bij} \simeq \mathsf{FinSet}$. In particular, there is a family $\mathsf{El} : \mathsf{Bij} \to \mathsf{Type}$ in which $\mathsf{El}(b)$ is the type underlying the finite set that each $b : \mathsf{Bij}$ corresponds to. □

With this replacement at hand, we can now define a small type of finite bags:

Definition 2.3.7. The type of finite bags is given by

$$\mathsf{Bag} : \mathsf{Type} \to \mathsf{Type}$$
$$\mathsf{Bag}(X) :\equiv \sum_{b:\mathsf{Bij}} (\mathsf{El}(b) \to X)$$ ⌟

The equivalence of proposition 2.3.6 extends to an equivalence $\mathsf{Bag}(X) \simeq \mathsf{Tote}(X)$ for any $X : \mathsf{Type}$. Together with theorem 2.3.4, we conclude that truncation of Bag is equal to the finite multiset *functor*:

Problem 2.3.8. Extend $\lambda X.\, \|\mathsf{Bag}(X)\|_0$ to an endofunctor $\|\mathsf{Bag}\|_0 : \mathsf{hSet} \to \mathsf{hSet}$.

Construction. The functorial action on maps is given by postcomposition under the truncation: Given $f : X \to Y$, define

$$\|\mathsf{Bag}\|_0(f) : \|\mathsf{Bag}(X)\|_0 \to \|\mathsf{Bag}(Y)\|_0$$
$$\|\mathsf{Bag}\|_0(f) :\equiv \lambda|(b, v)|_0.\, |(b, f \circ v)|_0$$ ⌟

Proposition 2.3.9. There is an equivalence $\|\mathsf{Bag}(X)\|_0 \simeq \mathsf{FMSet}(X)$, natural in X. Hence, $\|\mathsf{Bag}\|_0 = \mathsf{FMSet}$ as endofunctors of hSet.

Proof. We obtain the desired equivalence by combining $\mathsf{Bij} \simeq \mathsf{FinSet}$ (proposition 2.3.6) and $\|\mathsf{Tote}(X)\|_0 \simeq \mathsf{FMSet}(X)$ (theorem 2.3.4). It remains to show that the latter is natural in X, which is straightforward. The functors are equal by univalence of hSet. □

To establish an equivalence $\mathsf{FMSet}(X) \simeq \|\mathsf{Tote}(X)\|_0$ (theorem 2.3.4), we choose to give explicit maps in either direction. Alternatively, we could have shown that the forward map is both an injection and surjection. Both of these are propositions, hence we can avoid using the derived recursion principle applied in diagram 2.3 — at the cost of losing the explicit description of its inverse. A third proof exists as a shadow of a universal property: In proposition 2.1.14 we showed that $\mathsf{FMSet}(X)$ is (equal to) the free commutative monoid on X, $\mathsf{FCM}(X)$. In the same way, Piceghello shows that $\mathsf{Tote}(X)$ is equivalent to the *free symmetric monoidal groupoid* on X [Pic21, Corollary 5.103]. By set-truncating Tote, we truncate "morphism-sets" $xs =_{\mathsf{Tote}(X)} ys$ to a mere existence of permutations, hence equality of finite multisets. This equips $\|\mathsf{Tote}(X)\|_0$ with the structure of a (free) commutative monoid. In particular, $\mathsf{Tote}(1) \simeq \mathsf{FinSet} \simeq \mathsf{Bij}$, thus Bij presents the free symmetric monoidal groupoid on a single generator: objects are natural numbers $n, k, \ldots : \mathbb{N}$, whose morphism sets are isomorphisms of finite ordinals, $[n] \cong [k]$. The 2-path constructor seq ensures that the encoding $\mathsf{FinSet} \to \mathsf{Bij}$ is functorial. Proposition 2.3.6 shows that this is an equivalence of groupoids; Piceghello goes on to show this equivalence preserves the symmetric monoidal structure of either side. In particular, the coherences $\mathsf{hom}(\mathsf{id}) = \mathsf{refl}$ and $\mathsf{hom}(\alpha^{-1}) = \mathsf{hom}(\alpha)^{-1}$ are admissible.

2.3.3 The final Bag-coalgebra as a limit

Moving from sets to groupoids, we can present finite, unordered collections by a polynomial in groupoids; by definition, Bag is the extent of a $(1,0)$-truncated container:

$$\mathsf{Bag}(X) \simeq \sum_{b:\mathsf{Bij}} (\mathsf{El}(b) \to X) \simeq [\![\mathsf{Bij} \triangleleft \mathsf{El}]\!](X)$$

As such, theorem 2.3.1 immediately delivers a final Bag-coalgebra:

Corollary 2.3.10 (of theorem 2.3.1)**.** The pair $(\nu\mathsf{Bag}, \mathsf{out}_{\mathsf{Bag}})$ is a terminal object in the wild category of Bag-coalgebras. □

We would like to point out how neat this construction is: in § 2.2, we have struggled to prove that finite multisets *almost* admit a final coalgebra. Instead, theorem 2.3.1 establishes this for any wild endofunctor, as long as it is in the shape of a "polynomial" $\sum_{s:S}(P_s \to X)$ in Type. Its proof follows the intuition of the standard 1-categorical proof, but carefully reshapes types by equivalences as to not lose any coherence data.

At the cost of turning symmetries of finite sets into *data*, we get a concise and intuitive description of $\nu\mathsf{Bag}$: it consists of non-wellfounded trees, whose branches are indexed by finite sets. Identifications of such trees are characterized by a coinduction proof principle [ACS15, Theorem 18]: In order to show that $s =_{\nu\mathsf{Bag}} t$, it suffices to give a bisimulation R over $\nu\mathsf{Bag}$ such that $R(s,t)$.

Just like in § 2.2.3, we would like to understand to which degree $\nu\mathsf{Bag}$ induces a final coalgebra of finite *multisets*: Is $\|\nu\mathsf{Bag}\|_0$ the carrier of an M-coalgebra? If so, is it final? At least for finite approximations, $\|\nu\mathsf{Bag}\|_0$ and $\nu\mathsf{M}$ coincide:

Proposition 2.3.11. For all $n : \mathbb{N}$, there is an equivalence $\|\mathsf{Bag}^n(1)\|_0 \simeq \mathsf{M}^n(1)$.

Proof. The case $n \doteq 0$ reduces to $\|1\|_0 \simeq 1$. In the successor-case, proposition 2.3.9 tells us that

$$\|\mathsf{Bag}(\mathsf{Bag}^n(1))\|_0 \simeq \mathsf{M}(\mathsf{Bag}^n(1))$$

Furthermore, M is invariant under set truncation (theorem 2.1.13), hence

$$\simeq \mathsf{M}\|\mathsf{Bag}^n(1)\|_0$$

From the induction hypothesis, we conclude that

$$\simeq \mathsf{M}(\mathsf{M}^n(1))$$ □

Indeed, a similar argument shows that $\|\nu\mathsf{Bag}\|_0$ is a fixpoint of M. Unlike theorems 2.2.15 and 2.2.29, this does not depend on any non-constructive assumptions.

Theorem 2.3.12. There is an equivalence $\mathsf{fix} : \|\nu\mathsf{Bag}\|_0 \simeq \mathsf{M}\|\nu\mathsf{Bag}\|_0$.

Proof. Since $\nu\mathsf{Bag}$ is the final Bag-coalgebra (corollary 2.3.10), it is necessarily a Bag-fixpoint, hence

$$\|\nu\mathsf{Bag}\|_0 \simeq \|\mathsf{Bag}(\nu\mathsf{Bag})\|_0$$

The functors $\|\mathsf{Bag}\|_0$ and M coincide (proposition 2.3.9), hence

$$\simeq \mathsf{M}(\nu\mathsf{Bag})$$

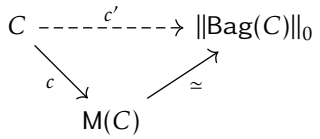
Invariance of M under set-truncation (theorem 2.1.13) then implies

$$\simeq \mathsf{M}\|\nu\mathsf{Bag}\|_0$$ □

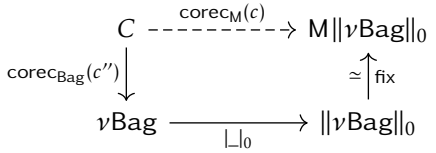
Unfortunately, obtaining a *largest* M-fixpoint appears to be, yet again, impossible without further assumptions. Furthermore, since we are attempting to construct a set-based fixpoint from a groupoid-based definition, it seems necessary to invoke two higher choice principles — one for sets, and one for groupoids, as given by definition 1.4.16:

Theorem 2.3.13. If $\mathsf{AC}_{0,\text{-}1}$ and $\mathsf{AC}_{1,0}$ hold, $(\|\nu\mathsf{Bag}\|_0, \mathsf{fix})$ is the final M-coalgebra.

Proof. Let $C : \mathsf{hSet}$ and $c : C \to \mathsf{M}(C)$. Our goal is to exhibit a (unique) coalgebra morphism $\mathsf{corec}_{\mathsf{M}}(c) : C \to \|\nu\mathsf{Bag}\|_0$. Consider the composite



The codomain of c' is a set-truncation of a groupoid, hence $\mathsf{AC}_{1,0}$ yields some $x : \|C \to \mathsf{Bag}(C)\|_0$. By induction on x, it suffices to construct $\mathsf{corec}_{\mathsf{M}}(c)$ from some $c'' : C \to \mathsf{Bag}(C)$. The latter is a Bag-coalgebra, hence we use the Bag-corecursor to define



It follows directly that $\mathsf{corec}_{\mathsf{M}}(c)$ is a coalgebra morphism. It remains to show that it is the unique such morphism. We lift this from the uniqueness of $\mathsf{corec}_{\mathsf{Bag}}(c'')$ as a Bag-coalgebra by an application of the axiom of choice for sets, $\mathsf{AC}_{0,-1}$. □

2.4 Conclusion

We have shown that the traditional construction of non-wellfounded, unordered trees is constructively taboo. Allthough we can construct such trees as the limit of some ω-chain, the finite multiset functor preserves this limit if and only if the constructively non-derivable LLPO holds. We believe that this a necessary limitation of set-based non-wellfounded constructions, and conjecture that, if M admits a final coalgebra (Z, ζ) (not necessarily obtained from a ω-chain), then $\mathsf{pres}_{\mathsf{M}}$ is injective, and LLPO holds.

If, however, we treat the symmetries of such trees as data, we obtain a well-behaved fixpoint: finite multisets are the shadow of a polynomial in h-groupoids, and this polynomial admits a (wild) final coalgebra almost for free. This translation from sets to groupoids works for a wide class of functors. Assume $F : \mathsf{hSet} \to \mathsf{hSet}$ is a sum of quotients of representables, i.e. of the shape

$$F(X) \simeq \sum_{s:S} (P_s \to X) / \sim_s$$

for some S, $P : S \to \mathsf{hSet}$ and a suitable relation $\sim_s$. In chapter 3, we are going to construct a polynomial

$$G(X) :\equiv \sum_{b:B} (\mathsf{El}(b) \to X)$$

in groupoids such that G is related to F in the same way that Tote is related to FMSet. In particular, we will show that F is a set-truncation of G, generalizing proposition 2.3.9.

3 Containers, from Sets to Groupoids

In the previous chapter we have seen that set quotients are unfit to describe non-wellfounded data in a constructive setting. Nonetheless, their induction principle and computation rules let us easily reason about such structures. In this chapter we will generalize the idea behind the identification $\mathsf{FMSet} \simeq \|\mathsf{Bag}\|_0$ (proposition 2.3.9), and give an interpretation of a much wider class of "unordered collections" in h-groupoids.

The central idea behind this translation is the following: Bag behaves nicely because it is a polynomial in h-groupoids, and its data is represented by a $(1,0)$-truncated container, i.e. a *symmetric container* [Gyl11]. On the other hand, FMSet is not a polynomial: instead of being a sum of representables, it is a sum of *quotients* of representables. For a reasonable class of quotients, such functors are represented exactly by so-called *quotient containers*, due to Abbott et al. [Abb+04]. In a quotient container, positions are invariant under a specified subgroup of their group of permutations, which induces a quotient in their extent.

In the first part of this chapter (§ 3.1), we show that it is easily possible to translate *any* quotient container to a symmetric container (theorem 3.1.25). This reinterprets the container in h-groupoids, promoting symmetries to data. It particular, it avoids constructive taboos when constructing the final coalgebra of its extension, as long as we do so in h-groupoids. We ensure that this lifting respects extensions: Generalizing proposition 2.3.9, we prove that the extent of the lifting agrees with the original extent, if truncated to a set-endofunctor.

Even though this translation from one kind of container to another is well-behaved, it does not obviously relate the *categories* of such containers. After all, we are comparing a 1-category to the 2-category of symmetric containers. Even if we were to adjust the types to match, we notice that the translation is not functorial — morphisms of quotient containers fail to preserve certain symmetries, preventing us from lifting them to morphisms of symmetric containers.

In § 3.2, we introduce an intermediate notion of container, slightly generalizing quotient containers. These *action containers* form a 1-category in which morphisms *do* preserve such symmetries. Among other things, this category of

containers models at least basic type formers — we show that it is closed under arbitrary sums and products by exhibiting it as the free coproduct completion of a category of group actions. Furthermore, constant action containers are exponentiable. As such, action containers represent a non-recursive fragment of strictly positive types, while at the same time encoding many types with symmetries.

In § 3.3.3, we equip action containers with the structure of a 2-category in which morphisms are related by a notion of "conjugator". This 2-category similarly arises from families of group actions, which lets us prove — in small steps, using the machinery of displayed bicategories [Ahr+21] — that the translation into symmetric containers is a locally fully-faithful 2-functor. This means that, not only do action containers correspond to certain symmetric containers, but so do their morphisms: conjugators relate morphisms of action container exactly when they are identified as morphisms of symmetric containers.

In § 3.4, we take a step back and ask whether a 2-categorical approach is truly necessary. We show that action containers are not symmetric containers with a certain property, but rather *structured*. We define a 2-category of *skeletal* symmetric containers, and prove that this extra structure forces the 2-category to be locally thin. This defines a 1-category of groupoid-based containers, and we prove that it is equivalent to the 1-category of action containers. We use this alternative presentation as structured symmetric containers to investigate *substitution* of such containers. We conclude that, in general, it seems difficult to prove that action containers (hence quotient containers) are closed under substitution, but we give some sufficient conditions on containers F and G for which $F[G]$ is well-defined.

3.1 Quotient- and symmetric containers

Both quotient- and symmetric containers represent datatypes with additional symmetries. In chapter 2, we have seen how to express finite multisets first as a quotient container, and then how to translate it into a symmetric container. In this section, we will show that it is not much harder to translate *any* quotient container into a symmetric one. Even though this construction is straightforward, it breaks down when one attempts to extend it to morphisms: In general, it is not possible to lift a morphism of quotient containers to one of symmetric containers, as the former truncates evidence on how symmetries are preserved.

3.1.1 Quotient containers

Quotient containers were introduced by Abbott et al. ([Abb+04]) as a way to add symmetries to containers in an extensional type theory with quotient types.

Since we have access to set quotients, we can express quotient containers directly in our type theory.

Definition 3.1.1. A *quotient container* $(S \triangleleft P /_\iota G)$ consists of

- a set of *shapes* S,
- a family of *positions* $P : S \to \mathsf{hSet}$,
- subgroups of *permissible permutations* $G : S \to \mathsf{Group}$,
- given by a family of embeddings of groups, $\iota : \prod_{s:S} G_s \hookrightarrow \Sigma(P_s)$. ⌟

Each embedding ι_s ensures that G_s is a subgroup of $\Sigma(P_s)$, the group of symmetries of $P(s)$. Whenever unambiguous, we elide ι and simply write $(S \triangleleft P/G)$. Similarly, each $g : G_s$ represents a permutation $\iota_s(g) : P_s \simeq P_s$, and we write g for its underlying map $P_s \to P_s$. Whenever we refer to the "symmetries of a container" we mean its collection of permissible permutations.

Like an ordinary container, a quotient container defines an *extension* as an endofunctor on the category of sets. Whereas for ordinary containers this is a polynomial functor, for quotient containers it is a sum of quotients of representables.

Definition 3.1.2. The extension of $(S \triangleleft P /_\iota G)$ is the map $[\![S \triangleleft P /_\iota G]\!]_/ : \mathsf{hSet} \to \mathsf{hSet}$ given by

$$[\![S \triangleleft P /_\iota G]\!]_/(X) \coloneqq \sum_{s:S} \frac{P_s \to X}{\sim_s}, \qquad v \sim_s w \coloneqq \exists g : G_s.\, v = w \circ \iota_s(g)$$ ⌟

Again, we call an endofunctor $F : \mathsf{hSet} \to \mathsf{hSet}$ a *quotient container functor* if it is equal to the extension of a quotient container, i.e. when $\mathsf{fiber}_{[\![\_]\!]_/}(F)$ is inhabited. The extensions of a quotient container whose positions are finite sets are exactly Joyal's *analytic functors* [Joy86].

Using quotient containers, it is easy to represent unordered collections.

Example 3.1.3. The quotient container of *unordered n-tuples* $\mathsf{U}_n \coloneqq (1 \triangleleft [n] /_\iota \Sigma_n)$ has a single shape. Over this shape, positions are the standard n-element set $[n]$. Any permutation of these n positions is permissible; $\iota_n : \Sigma_n \hookrightarrow \Sigma_n$ is the identity embedding. We call U_1 the *identity container*; it has a single shape, on which the trivial group acts.

The extension of U_n is the type of unordered n-tuples:

$$[\![\mathsf{U}_n]\!]_/(X) \doteq \sum_{\_:1} ([n] \to X)/\sim = X^n/\sim_{\Sigma_n}$$

where $x \sim_{\Sigma_n} y$ if and only if $x_i = y_{\sigma(i)}$ for some permutation $\sigma : \Sigma_n$. When $n = 1$, we obtain the identity function $[\![\mathsf{U}_1]\!]_/ X = X$. ⌟

Example 3.1.4. Finite multisets are a quotient container functor: FMSet (definition 2.1.11) is the extension of $((n : \mathbb{N}) \triangleleft [n] / \Sigma_n)$:

$$[\![(n : \mathbb{N}) \triangleleft [n] / \Sigma_n]\!]_/(X) \doteq \sum_{n:\mathbb{N}} ([n] \to X) / \sim_{\Sigma_n} = \mathsf{FMSet}(X)$$ ⌟

Intuitively, morphisms of quotient containers should again map shapes to shapes, and positions to positions. However, they should do so in a way that accounts for the permutation of positions. As a first approximation, we recall the notion of a premorphism:

Definition 3.1.5. A *premorphism* $(u \triangleleft f) : (S \triangleleft P / G) \rightharpoonup (T \triangleleft Q / H)$ of quotient containers consists of:

- a map of shapes $u : S \to T$,
- a map of positions $f : \prod_{s:S} Q_{us} \to P_s$, and
- a proof that f *preserves symmetries*, $\prod_{s:S} \prod_{g:G_s} \exists_{h:H_{us}} g \circ f_s = f_s \circ h$. ⌟

Morphisms are now defined up to a permutation of position-maps, i.e. as equivalence classes of premorphisms:

Definition 3.1.6. The type of morphisms $F \to G$ of quotient containers is the set quotient of $F \rightharpoonup G$ by the relation

$$(u \triangleleft f) \approx (u' \triangleleft f') \coloneqq \sum_{p:u=u'} f \approx'_p f'$$

in which $\_ \approx'_p \_ : (\prod_s Q_{us} \to P_s) \to (\prod_s Q_{u's} \to P_s) \to \mathsf{Type}$ is defined, by path-induction on $p : u = u'$, and relates two maps of positions whenever they differ by a permissible permutation:

$$f \approx'_{\mathsf{refl}} f' \coloneqq \forall s : S \exists h : H_{us}.\, f_s = f'_s \circ h$$ ⌟

By induction, it is easy to verify that morphisms of quotient containers compose appropriately:

Problem 3.1.7. Quotient containers form a category QuotCont.

Moreover, defining morphisms as quotients of premorphisms ensures that quotient containers represent a subcategory of endofunctors:

Proposition 3.1.8 ([Abb+04])**.** Extension is a full and faithful functor

$$[\![\_]\!]_/ : \mathsf{QuotCont} \to \mathsf{Endo}(\mathsf{hSet})$$

whose action on premorphisms $[\![[u, f]]\!]_{/X} : [\![F]\!]_/(X) \to [\![G]\!]_/(X)$ is given by

$$[\![[u, f]]\!]_{/X}(s, [v]) \coloneqq (s, [v \circ f])$$

is well-defined on morphisms. □

3.1.2 Symmetric containers

Symmetric containers were introduced by Gylterud to model types with symmetries by means of a groupoids. In [Gyl11], they are defined as a special case of a kind of container indexed by a *category* of shapes: there, they consist of a (categorical) groupoid of shapes, over which positions are a functor into sets. As such, symmetric containers internalize nicely as a certain kind of truncated container:

Definition 3.1.9. A symmetric container is a $(1,0)$-truncated container: its shapes form a h-groupoid, and positions are a family of h-sets. We denote the type of symmetric containers by $\mathsf{SymmCont} := (1,0)\text{-}\mathsf{Cont}$. ⌟

Whereas ordinary set-truncated containers form a 1-category, symmetric containers sit one step higher on the ladder of higher categories:

Problem 3.1.10. Symmetric containers form the objects of a 2-category. Morphisms are container morphisms, and 2-cells are homotopies thereof. This defines a $(2,1)$-category SymmCont, as 2-cells are invertible.

Construction. The original construction [Gyl11, Definition 2.1.2] translates without problems. In fact, the 2-dimensional coherences hold automatically as 2-cells are now simply identifications of 1-cells, not just natural transformations; for details see our implementation in Agda, [IIa, `SymmContCat`]. ⌟

Additionally, extension defines a 2-functor, embedding symmetric containers into pseudofunctors of h-groupoids, in the sense of proposition 1.5.18.

Proposition 3.1.11 ([Gyl11, Theorem 2.2.1])**.** Extension of symmetric containers defines a 2-functor $[\![\_]\!] : \mathsf{SymmCont} \to \mathsf{Endo}(\mathsf{hGrpd})$ that is locally an equivalence.

One advantage of internalizing symmetric containers as h-groupoids is that we are free to define groupoids of shapes as higher inductive types, encoding the desired symmetries directly in their constructors. For example, cyclic lists can be described using the symmetries of the circle, S^1:

Example 3.1.12. The symmetric container Cyc represents *cyclic lists*:

- Shapes are $\mathsf{Cyc}_{\mathsf{Sh}} := \mathbb{N} \times S^1$, a countable sum of circles. The index $\mathbb{N}$ encodes the length of a list; loops $(n, \mathsf{base}) =_{\mathbb{N}\times S^1} (n, \mathsf{base})$ represent possible cyclic shifts of positions.
- Positions $\mathsf{Cyc}_{\mathsf{Ps}} : \mathbb{N} \times S^1 \to \mathsf{hSet}$ are defined by induction on S^1: Over the n-th circle we have n distinct positions, $\mathsf{Cyc}_{\mathsf{Ps}}(n, \mathsf{base}) := [n]$. On the loop, positions identify finite sets by a cyclic shift, $\mathsf{Cyc}_{\mathsf{Ps}}(\mathsf{refl}_n, \mathsf{loop}) := \mathsf{ua}(\mathsf{suc}_n)$ where $\mathsf{suc}_n : [n] \simeq [n]$ is the *successor equivalence* $\mathsf{suc}_n(i) := (i+1) \mod n$.

Its extension is $[\![\mathsf{Cyc}]\!](X) \simeq \sum_{n:\mathbb{N}} \sum_{x:S^1} \mathsf{Cyc}_{\mathsf{Ps}}(n,x) \to X$. We can identify n-tuples $v, w : [n] \to X$ in $[\![\mathsf{Cyc}]\!](X)$ if they are equal up to a cyclic permutation of their domain $[n]$: if there exists some $k : \mathbb{Z}$ such that $v = w \circ \mathsf{suc}_n^k$, then univalence induces a path $p : v =^B_{\mathsf{loop}^k} w$ over the family $B(x : S^1) :\equiv \mathsf{Cyc}_{\mathsf{Ps}}(n,x) \to X$. The triple of paths $\mathsf{refl}_n : n = n$, $\mathsf{loop}^k : \mathsf{base} = \mathsf{base}$, and p together define an identification $(n, \mathsf{base}, v) =_{[\![\mathsf{Cyc}]\!](X)} (n, \mathsf{base}, w)$. In fact, this yields an equivalence characterizing identifications

$$\prod_{v,w:[n]\to X} \Big((n, \mathsf{base}, v) =_{[\![\mathsf{Cyc}]\!](X)} (n, \mathsf{base}, w)\Big) \simeq \sum_{k:\mathbb{Z}} v = w \circ \mathsf{suc}_n^k$$

via the well-known equivalence $\Omega(S^1, \mathsf{base}) \simeq \mathbb{Z}$ [Uni13, Corollary 8.1.10]. Note that this notion of cyclic list remembers the number of "rounds" a cyclic shift has taken. If v and w are identical up to shift by k positions, then they are also identical if shifted by $k + n$ positions (one additional round of rotation). Both $p : v = w \circ \mathsf{suc}_n^k$ and $p' : v = w \circ \mathsf{suc}_n^{k+n}$ yield different paths in $[\![\mathsf{Cyc}]\!](X)$. ⌟

3.1.3 Delooping of groups

In order to create a symmetric container from a quotient container, we have to come up with a *groupoid* of shapes that encodes the symmetry groups of the quotient container. Any group can of course be seen as the isomorphism set of a single-object (categorical) groupoid. We can express this coherently in a univalent setting: any group has a unique *delooping* — a pointed, connected (homotopy) groupoid whose fundamental group is isomorphic to the group itself. The delooping is an instance of an Eilenberg–MacLane space, i.e. a type with trivial homotopy groups in all but one degree. Such spaces have been studied as higher inductive types, and we present the delooping of a group such that it coincides with the HIT $K(G,1)$ of [LF14].

Definition 3.1.13. The *delooping* of a group G is the higher inductive type $\mathbf{B}G$ given by the point- and path constructors

$$\frac{}{\bullet : \mathbf{B}G} \qquad \frac{g : G}{\mathsf{loop} : \bullet = \bullet} \qquad \frac{g, h : G}{\mathsf{comp} : \mathsf{loop}(g) \cdot \mathsf{loop}(h) = \mathsf{loop}(gh)}$$

together with a constructor asserting that $\mathbf{B}G$ is a groupoid. Its *dependent eliminator* is, for any family $B : \mathbf{B}G \to \mathsf{hGrpd}$, a map

$$\frac{b_0 : B(\bullet) \quad \varphi : \prod_{g:G} b_0 =^B_{\mathsf{loop}(g)} b_0 \quad \varphi\text{-}\mathsf{comp} : \prod_{g,h:G} \varphi(g) \cdot \varphi(h) =^B_{\mathsf{comp}(g,h)} \varphi(gh)}{\mathsf{elim}_B : \prod_{x:\mathbf{B}G} B(x)}$$

with the expected computation rules. For constant families $\lambda\_.\, X$, this simplifies to the *recursor*

$$\frac{x_0 : X \quad \varphi : G \to X = X \quad \varphi\text{-comp} : \prod_{g,h:G} \varphi(g) \cdot \varphi(h) = \varphi(gh)}{\mathsf{rec}_X : \mathbf{B}G \to X}$$ ⌟

As expected for an inductive type, its eliminator fully characterizes dependent functions out of it.

Proposition 3.1.14. For all groups G and families $B : \mathbf{B}G \to \mathsf{hGrpd}$, currying elim_B induces an equivalence

$$\left(\sum_{b_0:B(\bullet)} \sum_{\varphi:\prod_{g:G} b_0 =^B_{\mathsf{loop}(g)} b_0} \prod_{g,h:G} \varphi(g) \cdot \varphi(h) =^B_{\mathsf{comp}(g,h)} \varphi(gh)\right) \simeq \prod_{x:\mathbf{B}G} B(x)$$

If B is of a lower truncation level, we additionally have the following:

1. If B is a family of sets, then

$$\left(\sum_{b_0:B(\bullet)} \prod_{g:G} b_0 =^B_{\mathsf{loop}(g)} b_0\right) \simeq \prod_{x:\mathbf{B}G} B(x)$$

2. If B is a family of propositions, then

$$B(\bullet) \simeq \prod_{x:\mathbf{B}G} B(x)$$

Recall that the loop space $\Omega(X, x_0) \doteq (x_0 = x_0)$ of any pointed h-groupoid (X, x_0) forms a group under composition of paths. From this perspective, the non-dependent recursion principle says that maps out of $\mathbf{B}G$ are exactly given by a group homomorphism:

Lemma 3.1.15. Maps $\mathbf{B}G \to X$ into an h-groupoid X are uniquely determined by a point of X and a *group homomorphism*: rec_X induces an equivalence

$$\left(\sum_{x_0:X} G \dashrightarrow \Omega(X, x_0)\right) \simeq (\mathbf{B}G \to X)$$

The justifies defining maps using pattern-matching notation

$$\begin{aligned} &f : \mathbf{B}G \to X \\ &f(\bullet) \coloneqq x_0 \\ &f(\mathsf{loop}(g)) \coloneqq \varphi(g) \end{aligned}$$

whenever φ is a group homomorphism of appropriate type. □

A simple *encode-decode* argument (in the sense of [Uni13, §8.9]) shows that the constructor loop is an equivalence of groups, i.e. a group homomorphism that is an equivalence of the underlying carriers.

Lemma 3.1.16. For any group G, the constructor loop is an equivalence of groups G and $\Omega(\mathbf{B}G, \bullet)$. It arises as the preimage $\mathrm{rec}^{-1}_{\mathbf{B}G}(\mathrm{id}_{\mathbf{B}G}) = (\bullet, \mathsf{loop})$. In particular, the coherences

$$\mathsf{loop}(1_G) = \mathsf{refl} \qquad \text{and} \qquad \forall g : G.\ \mathsf{loop}(g^{-1}) = \mathsf{loop}(g)^{-1}$$

are admissible. □

This implies that any group homomorphism lifts to a map of h-groupoids:

Problem 3.1.17. Given any group homomorphism $\varphi : G \dot{\to} H$, define a map

$$\begin{aligned} &\mathbf{B}\varphi : \mathbf{B}G \to \mathbf{B}H \\ &\mathbf{B}\varphi(\bullet) :\equiv \bullet \\ &\mathbf{B}\varphi(\mathsf{loop}(g)) :\equiv \mathsf{loop}(\varphi(g)) \end{aligned}$$

Construction. By recursion (lemma 3.1.15), its suffices to show that the composite $\mathsf{loop} \circ \varphi$ is a group homomorphism: φ is by assumption, and loop by lemma 3.1.16. ⌟

Since group actions are in particular homomorphisms, the above associates to any action of G a family $\mathbf{B}G \to \mathsf{hSet}$:

Definition 3.1.18. Let $\sigma : G \dot{\to} \Sigma(X)$ be an action of G on X. Its *associated bundle* is

$$\begin{aligned} &\bar{\mathbf{B}}\sigma : \mathbf{B}G \to \mathsf{hSet} \\ &\bar{\mathbf{B}}\sigma(\bullet) :\equiv X \\ &\bar{\mathbf{B}}\sigma(\mathsf{loop}(g)) :\equiv \mathsf{ua}(\sigma(g)) \end{aligned}$$ ⌟

If a group acts faithfully on a set, then its associated bundle is an embedding on path spaces:

Proposition 3.1.19. If $\sigma : G \dot{\to} \Sigma(X)$ acts faithfully, then $\bar{\mathbf{B}}\sigma : \mathbf{B}G \to \mathsf{hSet}$ is a set-truncated map.

Proof. By lemma 1.4.3, it suffices to show that $\mathsf{cong}_{\bar{\mathbf{B}}\sigma} : x = y \to \bar{\mathbf{B}}\sigma(x) = \bar{\mathbf{B}}\sigma(y)$ is an embedding for all $x, y : \mathbf{B}G$. This is a proposition, so by induction on x and

y it is enough to establish this whenever $x \doteq y \doteq \bullet$ (cf. propositions 3.1.14.(2) and 3.1.14). The computation rules for $\bar{\mathbf{B}}\sigma$ yield a commutative square

$$\begin{array}{ccc} G & \xrightarrow{\mathsf{loop}} & (\bullet = \bullet) \\ {\scriptstyle\sigma}\downarrow & & \downarrow{\scriptstyle \mathsf{cong}_{\bar{\mathbf{B}}\sigma}} \\ \Sigma X & \xrightarrow[\mathsf{ua}]{} & (X = X) \end{array}$$

The top- and bottom maps are equivalences by lemma 3.1.16 and univalence, respectively. On the left, σ is an embedding by assumption. Therefore, $\mathsf{cong}_{\bar{\mathbf{B}}\sigma}$ must be an embedding. □

Similarly, we can use the associated bundle to encode the set of *orbits* (definition 1.7.7) of an action: it is equivalent to the connected components of the total space of its associated bundle.

Lemma 3.1.20. For all G acting on X via σ, we have $\|\sum_{x:\mathbf{B}G} \bar{\mathbf{B}}\sigma(x)\|_0 \simeq X/\sigma$.

Proof. We provide an explicit isomorphism. In the left-to-right direction, the codomain is a set. By recursion on the truncation, it suffices to give a map $f : \prod_{x:\mathbf{B}G} \bar{\mathbf{B}}\sigma(x) \to X/\sigma$. On the point, the surjection onto the set quotient of orbits defines $f(\bullet) \coloneqq [\_] : X \to X/\sigma$. It remains to show this is well-defined on loops, which reduces to showing that $\prod_{g:G}\prod_{x:X}[x] = [\sigma_g(x)]$. This holds since $x \sim_\sigma \sigma_g(x)$ by definition of the orbit-relation.

The inverse map $h : X/\sigma \to \|\sum_{x:\mathbf{B}G} \bar{\mathbf{B}}\sigma(x)\|_0$ is defined by recursion on X/σ. Let $h([x]) \coloneqq |(\bullet, x)|_0$, and show that $x \sim_\sigma y$ implies $h([x]) = h([y])$. Our goal is a proposition, so we can assume that we are given some $g : G$ such that $p : \sigma_g(x) = y$. From this, we construct a path $(\bullet, x) = (\bullet, y)$ in the total space of the associated bundle: the first component is given by $\mathsf{loop}(g) : \bullet = \bullet$, the second by a dependent path $x =_{\mathsf{ua}(\sigma_g)} y$, obtained by transporting along p.

Verifying that the above maps are mutual inverses is straightforward: after applying the eliminators into propositions of $\mathbf{B}G$ and X/σ, respectively, the desired equalities hold trivially. □

3.1.4 Lifting quotient containers

Applying the above construction to all symmetries of a quotient container, we obtain a symmetric container:

Definition 3.1.21. The *delooping* of a quotient container $F \doteq (S \triangleleft P /_l G)$ is the symmetric container $\mathbf{B}F \coloneqq (\mathbf{S} \triangleleft \mathbf{P})$ consisting of

$$\begin{array}{ll} \mathbf{S} : \mathsf{hGrpd} & \mathbf{P} : \mathbf{S} \to \mathsf{hSet} \\ \mathbf{S} \coloneqq \sum_{s:S} \mathbf{B}G_s \quad \text{and} & \mathbf{P}(s, x) \coloneqq \bar{\mathbf{B}}\iota_s(x) \end{array} \quad ⌟$$

Recall that for a pointed type (X, x), its set of connected components is $\pi_0(X) \coloneqq \|X\|_0$, and its fundamental group is the set of loops $\pi_1(X, x) \coloneqq \|\Omega(X, x)\|_0$. We think of the shapes $\mathbf{S}$ as having one point for each shape $s : S$, surrounded by loops given by elements in G_s, as illustrated by the example in figure 3.1.

Proposition 3.1.22. Let $F \doteq (S \triangleleft P \,/\, G)$. With $\mathbf{S}$ and $\mathbf{P}$ as above, we have

$$\pi_0(\mathbf{S}) \simeq S \qquad \text{and} \qquad \pi_1(\mathbf{S}, (s, \bullet)) \simeq G_s.$$

Proof. By definition, the connected components of $\mathbf{S}$ are

$$\pi_0(\mathbf{S}) \simeq \left\| \textstyle\sum_{s:S} \mathbf{B}G_s \right\|_0$$

Since S is a set, we can smuggle the truncation under the sum,

$$\simeq \sum_{s:S} \|\mathbf{B}G_s\|_0$$

But $\mathbf{B}G_s$ is a *connected* type, hence its set-truncation is contractible. Therefore, $\pi_0(\mathbf{S}) \simeq S$. Similarly, its fundamental group around $s : S$ and $\bullet : \mathbf{B}G_s$ is

$$\begin{aligned} \pi_1(\mathbf{S}, (s, \bullet)) &\simeq \|(s, \bullet) = (s, \bullet)\|_0 \\ &\simeq (s, \bullet) = (s, \bullet) \\ &\simeq \sum_{p : s = s} \bullet =_p \bullet \end{aligned}$$

Again, S is a set, so $s = s$ is contractible:

$$\begin{aligned} &\simeq (\bullet = \bullet) \\ &\simeq \Omega(\mathbf{B}G_s, \bullet) \\ &\simeq G_s \end{aligned}$$

The last step is lemma 3.1.16. □

By definition, symmetries of a quotient container are subgroups of permutations of positions, hence they act *faithfully*. This is reflected in the positions of its delooping, which becomes an embedding of path spaces.

Proposition 3.1.23. The family $\mathbf{B}(F)_{\mathsf{Ps}} : \mathbf{B}(F)_{\mathsf{Sh}} \to \mathsf{hSet}$ is set-truncated for all quotient containers F.

Proof. Let $F \doteq (S \triangleleft P /_\iota G)$ and $\mathbf{S}, \mathbf{P}$ as above. For each $X : \mathsf{hSet}$, we have $\mathsf{fiber}_{\mathbf{P}}(X) \simeq \sum_{s:S} \mathsf{fiber}_{\tilde{\mathbf{B}}\iota_s}(X)$. The latter is a set: S is a set, and since we assume ι_s to be faithful, so are the fibers of $\tilde{\mathbf{B}}\iota_s$ by proposition 3.1.19. □

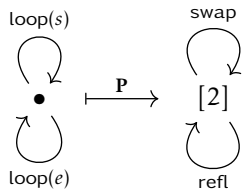


Figure 3.1: Delooping of the container $\mathsf{U}_2 \coloneqq (1 \triangleleft 2 /\!\!/ \mathbb{Z}_2)$ of unordered pairs. There are two symmetries in $\mathbb{Z}_2 = \{e, s \mid s^2 = e\}$: s represents identification of pairs by swapping of elements; e identification by "doing nothing". In $\mathbf{S}$, both are loops around $\bullet : \mathbf{B}\mathbb{Z}_2$. $\mathbf{P}$ sends $\mathsf{loop}(s)$ to $\mathsf{swap} : [2] = [2]$, the path induced by swapping the two elements of $[2]$. By lemma 3.1.16, it has to send $\mathsf{loop}(e)$ to refl.

This way of constructing a symmetric container is in some way conservative; the extensions of a quotient container agrees with the extensions of its delooping, if truncated to sets:

Problem 3.1.24. Given a quotient container F and a set X, define an equivalence

$$\|[\![\mathbf{B}F]\!](X)\|_0 \simeq [\![F]\!]_{/}(X)$$

Construction. Let $F \doteq (S \triangleleft P /\!\!/_\iota G)$. Unfold the definitions of $[\![\_]\!]$ and $\mathbf{B}$,

$$\|[\![\mathbf{B}F]\!](X)\|_0 \simeq \|\textstyle\sum_{s:S} \sum_{x:\mathbf{B}G_s} \bar{\mathbf{B}}\iota_s(x) \to X\|_0$$

and, as S is a set, move the truncation under the sum

$$\simeq \textstyle\sum_{s:S} \|\sum_{x:\mathbf{B}G_s} \bar{\mathbf{B}}\iota_s(x) \to X\|_0$$

Notice that G_s acts on the function type $P_s \to X$ via precomposition. This action, call it ι_s^*, has an associated bundle $\bar{\mathbf{B}}\iota_s^* : \mathbf{B}G_s \to \mathsf{hSet}$. It is easy to see that $\bar{\mathbf{B}}\iota_s^*(x) \simeq (\bar{\mathbf{B}}\iota_s(x) \to X)$, hence

$$\simeq \textstyle\sum_{s:S} \|\sum_{x:\mathbf{B}G_s} \bar{\mathbf{B}}\iota_s^*(x)\|_0$$

By lemma 3.1.20, the connected components of this bundle correspond to the set of ι_s^*-orbits, hence

$$\simeq \textstyle\sum_{s:S} (P_s \to X) / \iota_s^*$$

But this is exactly how extension of a quotient container is defined:

$$\simeq \textstyle\sum_{s:S} (P_s \to X) / \sim_s \simeq [\![F]\!]_{/}(X)$$ ⌟

For any symmetric container $(S \triangleleft P)$, the map $\|[\![S \triangleleft P]\!](\_)\|_0 : \mathsf{hSet} \to \mathsf{hSet}$ extends to an endofunctor on sets. As hinted at in § 2.4, this means that any quotient container F is related to its delooping $\mathbf{B}F$ in the same way that FMSet is related to Bag, generalizing proposition 2.3.9 to arbitrary quotient containers:

Theorem 3.1.25. For all $F : \mathsf{QuotCont}$, the equivalence of problem 3.1.24 is natural in X. Therefore, $\|[\![\mathbf{B}F]\!](\_)\|_0$ and $[\![F]\!]_/$ are propositionally equal as set-endofunctors. □

The equivalence constructed in problem 3.1.24 tells us that the extent map for quotient containers factors through symmetric containers; on objects, the diagram

$$\begin{array}{ccc} \mathsf{SymmCont} & \xrightarrow{[\![\_]\!]} & \mathsf{Endo(hGrpd)} \\ {\scriptstyle\mathbf{B}}\uparrow & & \downarrow{\scriptstyle \lambda F.\, \|F(\_)\|_0} \\ \mathsf{QuotCont} & \xrightarrow[[\![\_]\!]_/]{} & \mathsf{Endo(hSet)} \end{array}$$

commutes. It is tempting to ask whether this extends to an identification of functors typed $\mathsf{QuotCont} \to \mathsf{Endo(hSet)}$. However, as it is, this is not a well-posed question: neither SymmCont nor functors Endo(hGrpd) are types of objects of a 1-category, but rather those of $(2,1)$-categories. Even if they were,[1] it is not clear how $\mathbf{B}$ is supposed to act on morphisms of quotient containers. Given a premorphism $(u \triangleleft f) : (S \triangleleft P \,/\, G) \nrightarrow (T \triangleleft Q \,/\, H)$, we would have to provide a morphism between the lifted containers $\mathbf{B}(S \triangleleft P \,/\, G)$ and $\mathbf{B}(S \triangleleft P \,/\, G)$. In particular, we would have to give a function of shapes

$$\begin{aligned} \sum_{s:S} \mathbf{B}G_s &\to \sum_{t:T} \mathbf{B}H_t \\ (s, x) &\mapsto (u(s), ?) \end{aligned} \tag{3.1}$$

To define $? : \mathbf{B}G_s \to \mathbf{B}H_{us}$, it would suffice to provide a homomorphism $G_s \dot{\to} H_{us}$ (problem 3.1.17). However, we are not given this information: even though f preserves symmetries by definition of premorphisms, this only tells us that, for each $g : G_s$, there merely exists *some* $h : H_{us}$. Even if we had access to an explicit function $\varphi : G_s \to H_{us}$, it would not have to be a group *homomorphism*. In fact, it is easy to construct premorphisms not arising from group homomorphisms:

[1] One could, for example, replace both by their "homotopy categories" [Uni13, Example 9.1.18], which set-truncate their types of morphisms.

Example 3.1.26. Consider $(\mathsf{id} \triangleleft !) : \mathsf{U}_1 \nrightarrow \mathsf{U}_2$. The terminal map $! : [2] \to [1]$ trivially preserves symmetries: The diagram

$$\begin{array}{ccc} [2] & \xrightarrow{!} & [1] \\ {\scriptstyle \varphi(g)}\downarrow & & \downarrow{\scriptstyle g} \\ [2] & \xrightarrow[!]{} & [1] \end{array}$$

commutes for any choice of $g : \Sigma_1$ and $\varphi : \Sigma_1 \to \Sigma_2$, in particular $\varphi(\_) \coloneqq \mathsf{swap}$, which is *not* a group homomorphism. ⌟

Since morphisms of quotient containers are equivalence classes, it might be possible to find another premorphism in the same class for which this assignment is a homomorphism. In example 3.1.26, one could pick $\varphi(\_) \coloneqq \mathsf{id}$, which clearly is. Doing so however for *arbitrary* symmetry groups seems impossible without invoking some form of choice principle.

Instead, we are going to augment the definition of quotient containers with the necessary information, and investigate their relation to symmetric containers more closely.

3.2 Action containers

In this section, we define *action containers* and assemble them into a 1-category. Morphisms in this category are akin to premorphisms of quotient containers. In particular, they are not quotients by a relation on positions.

Unlike quotient containers, the symmetries of action containers are not limited to subgroups of permutations of positions. Instead, an action container has, for each shape, a chosen group *acting* on its set of positions. This lets us flexibly introduce symmetries, e.g. by letting an infinite group act directly on a finite set, instead of having to describe its image in a permutation group.

The category of action containers admits a number of limits and colimits. By presenting it as a category of families of group actions, we can easily read off the usual closure properties of containers with respect to products and coproducts. Additionally, constant action containers are exponentiable.

Definition 3.2.1. An *action container* $(S \triangleleft P \triangleright^{\sigma} G)$ consists of

- a set of *shapes* $S : \mathsf{hSet}$,
- a family of *positions* $P : S \to \mathsf{hSet}$, and
- a family of *symmetries* $G : S \to \mathsf{Group}$
- *acting* on the positions via $\sigma : \prod_{s:S} G_s \dashrightarrow \Sigma(P_s)$. ⌟

Action containers let us define containers from symmetries that do not act faithfully on positions:

Example 3.2.2 (cyclic lists)**.** We define an action container representing lists up to cyclic permutations. In the container $\mathrm{Cyc} \coloneqq (\mathbb{N} \triangleleft [\_] \triangleright^{\sigma} \mathbb{Z})$, the integers $\mathbb{Z}$ act on a finite set $[n]$ as follows: for each $n : \mathbb{N}$, the action $\sigma_n : \mathbb{Z} \to \Sigma_n$ maps $k : \mathbb{Z}$ to a cyclic shift of $[n]$ by k positions, that is $\sigma_n(k) \coloneqq \lambda i.\,(i + k) \mod n$. This action is not faithful; both $\sigma_n(k)$ and $\sigma_n(k + n)$ induce the same permutation of $[n]$. In the parlance of group theory, σ_n has non-trivial kernel, namely $n\mathbb{Z} = \{nk \mid k \in \mathbb{Z}\}$. ⌟

It is easy to specify action containers by means of $\mathbb{Z}$-actions: $\mathbb{Z}$ is the free group on a single generator, thus it suffices to define the action of $1 : \mathbb{Z}$. In example 3.2.2 it is enough to define $\sigma_n(1) = \lambda i.\,(i + 1) \mod n$, i.e. shifts by one position. This is impossible for quotient containers with finitely many positions; $\mathbb{Z}$ is simply never a subgroup of the (finite) group Σ_n.

To avoid pathologies such as example 3.1.26, we require morphisms to preserve the full structure of action containers. In particular, they should preserve symmetries in a way that is compatible with the actions. Recall from definition 1.7.8 that a morphism of G-sets $f : (P, \sigma) \to (Q, \tau)$ is a G-equivariant map, i.e. a map such that

$$\begin{array}{ccc} Q & \xrightarrow{f} & P \\ {\scriptstyle \tau(g)}\downarrow & & \downarrow{\scriptstyle \sigma(g)} \\ Q & \xrightarrow[f]{} & P \end{array}$$

commutes for all $g : G$. We define morphisms between action containers in terms of equivariant maps, accounting for the fact that positions on either side are acted upon by different groups. Instead of asserting equivariance with respect to a group, we require maps of positions to be equivariant with respect to group homomorphisms relating the containers.

Definition 3.2.3. A morphism $(u \triangleleft f \triangleright \varphi) : (S \triangleleft P \triangleright^{\sigma} G) \to (T \triangleleft Q \triangleright^{\tau} H)$ of action containers consists of

- a map of shapes $u : S \to T$,
- maps on positions $f : \prod_{s:S} Q_{us} \to P_s$, and
- a family of group homomorphisms $\varphi : \prod_{s:S} G_s \dot{\to} H_{us}$,

such that f is *equivariant* with respect to φ: the square

$$\begin{array}{ccc} Q_{us} & \xrightarrow{f_s} & P_s \\ {\scriptstyle \tau_{us}(\varphi_s(g))}\downarrow & & \downarrow{\scriptstyle \sigma_s(g)} \\ Q_{us} & \xrightarrow[f_s]{} & P_s \end{array}$$

commutes for all $s : S$ and $g : G_s$.

Given a morphism $e : F \to G$ of action containers, we write e_{Sh}, e_{Ps} and e_{Sm} for its action on shapes, positions and symmetries, respectively. ⌟

3.2.1 Action containers as families of actions

It is straightforward to verify that action containers form a category.

Proposition 3.2.4. Action containers and their morphisms form a category, ActionCont. □

Yet, the above is somewhat tedious as one has to carefully re-index maps of positions and symmetries over shape maps. Instead, we opt to present ActionCont as an equivalent category of *families of group actions*. This is obvious on the level of objects: over each shape $s : S$, the data of a container $(S \triangleleft P \triangleright^{\sigma} G)$ is exactly that of an action σ_s of G_s on P_s. We ensure that this analogy extends to morphisms, and explain how the *ad-hoc* definition of equivariance in definition 3.2.3 arises naturally from a notion of equivariance for actions. In particular, we consider the following:

Definition 3.2.5. Given actions $\sigma : G \dot{\to} \Sigma(P)$ and $\tau : H \dot{\to} \Sigma(Q)$, a map $f : Q \to P$ is equivariant over a homomorphism $\varphi : G \dot{\to} H$ if

$$\begin{array}{ccc} Q & \xrightarrow{f} & P \\ {\scriptstyle \tau(\varphi g)}\downarrow & & \downarrow{\scriptstyle \sigma(g)} \\ Q & \xrightarrow[f]{} & P \end{array}$$

commutes, that is, it satisfies

$$\mathsf{isEquivariant}_{\sigma,\tau}(f,\varphi) \coloneqq \prod_{g:G} \sigma(g) \circ f = f \circ \tau(\varphi(g)) \qquad \lrcorner$$

Remark 3.2.6. Any G-equivariant map $f : (Q,\tau) \to (P,\sigma)$ is equivariant over the identity homomorphism id_G in the sense of definition 3.2.5. This defines a functor $G\text{-}\mathsf{Set}^{\mathsf{op}} \to \mathsf{Action}$, exhibiting G-sets and their morphisms as actions over a fixed group. $\lrcorner$

With this, we consider a version of the category of G-sets in which equivariant maps are permitted to go between sets with actions of different groups.

Problem 3.2.7. Define a category Action, by displaying the following types of objects and morphisms over $\mathsf{Group} \times \mathsf{hSet}^{\mathsf{op}}$:

- Over $G : \mathsf{Group}_0$ and $P : \mathsf{hSet}_0^{\mathsf{op}}$, displayed objects are G-actions on P, i.e. group homomorphisms

$$\mathsf{Action}_0(G,P) \coloneqq (G \dot{\to} \Sigma(P))$$

- Displayed morphisms between $\sigma : \mathsf{Action}_0(G, P)$ and $\tau : \mathsf{Action}_0(H, Q)$ over $(\varphi : G \dot{\to} H, f : P \leftarrow Q)$ are proofs of the proposition "f is equivariant over φ":

$$\mathsf{Action}_1((\varphi, f); \sigma, \tau) \coloneqq \mathsf{isEquivariant}_{\sigma,\tau}(\varphi, f)$$

Construction. The identity function is clearly equivariant over the identity homomorphism. Diagrammatically, equivariant maps compose by horizontal pasting of proofs:

$$\begin{array}{ccc} P & \xleftarrow{f} & Q \\ {\scriptstyle \sigma(g)}\downarrow & & \downarrow{\scriptstyle \tau(\varphi g)} \\ P & \xleftarrow[f]{} & Q \end{array} \;;\; \begin{array}{ccc} Q & \xleftarrow{e} & R \\ {\scriptstyle \tau(h)}\downarrow & & \downarrow{\scriptstyle \upsilon(\psi h)} \\ Q & \xleftarrow[e]{} & R \end{array} \;\coloneqq\; \begin{array}{ccc} P & \xleftarrow{fe} & R \\ {\scriptstyle \sigma(g)}\downarrow & & \downarrow{\scriptstyle \upsilon(\psi\varphi g)} \\ P & \xleftarrow[fe]{} & R \end{array}$$

The displayed category is *thin* (isEquivariant is a proposition), so composition is automatically unital and associative. ⌟

We choose to display actions over $\mathsf{hSet}^{\mathsf{op}}$ to match the variance of position maps of containers. Even if displayed over hSet, actions would form a category; which one we pick is arbitrary, just as choosing between left- or right-actions is.

Next, we recall the definition of the category of families, which is most naturally presented as a displayed category:

Definition 3.2.8. Let $\mathcal{C}$ be a 1-category. The category $\mathsf{Fam}(\mathcal{C})$ is the total category of the following category displayed over hSet:

- Displayed objects are set-indexed families of $\mathcal{C}$-objects,

$$\begin{aligned} &\mathsf{Fam}(\mathcal{C})_0 : \mathsf{hSet}_0 \to \mathsf{Type} \\ &\mathsf{Fam}(\mathcal{C})_0(I) \coloneqq I \to \mathcal{C}_0 \end{aligned}$$

- Morphisms displayed over $u : I \to J$ are families of $\mathcal{C}$-morphisms,

$$\begin{aligned} &\mathsf{Fam}(\mathcal{C})_1(u; \_, \_) : (I \to \mathcal{C}_0) \to (J \to \mathcal{C}_0) \to \mathsf{hSet} \\ &\mathsf{Fam}(\mathcal{C})_1(u; X, Y) \coloneqq \textstyle\prod_{i:I} \mathcal{C}_1(X_i, Y_{ui}) \end{aligned}$$

Morphisms compose by pointwise composition in $\mathcal{C}$. ⌟

The category of action containers is equivalent to those of families of group actions:

Theorem 3.2.9. There is an equivalence of categories $\mathsf{ActionCont} \simeq \mathsf{Fam}(\mathsf{Action})$ whose action on objects is given by $(S \triangleleft P \triangleright^{\sigma} G) \mapsto (S, \lambda s.\, (G_s, P_s, \sigma_s))$. □

By breaking down the construction of ActionCont this way, we can now easily derive additional properties and structure.

Proposition 3.2.10. Action is a univalent category.

Proof. The base category $\mathsf{Group} \times \mathsf{hSet}^{\mathsf{op}}$ is univalent, so by proposition 1.5.14 it suffices to show that $\mathsf{Fib}_{\mathsf{Action}}(G, P)$ is univalent for all G and P. Morphisms in the fiber category encode equality of G-sets, that is

$$\mathsf{Fib}_{\mathsf{Action}}(G,P)(\sigma,\tau) \doteq \mathsf{isEquivariant}_{\sigma,\tau}(\mathsf{id}_P, \mathsf{id}_G) \simeq (\forall g.\, \sigma(g) = \tau(g)) \simeq (\sigma = \tau).$$

Both sides of the above are propositions, so the map underlying the right-to-left direction must be idToIso. This proves that $\mathsf{Fib}_{\mathsf{Action}}(G, P)$ is univalent. □

Proposition 3.2.11. Fam preserves univalence: if $\mathcal{C}$ is univalent, then so is $\mathsf{Fam}(\mathcal{C})$.

Proof. Again, we use proposition 1.5.14, and show that the fiber categories are univalent. For all $I : \mathsf{hSet}_0$, the category $\mathsf{Fib}_{\mathsf{Fam}(\mathcal{C})}(I)$ is propositionally equal to the I-fold product category $\mathcal{C}^I$, which is univalent if $\mathcal{C}$ is. □

Corollary 3.2.12. ActionCont is a univalent category.

Proof. Action is univalent (proposition 3.2.10), and so are families of actions (proposition 3.2.11). But univalence is preserved under equivalence of categories, hence ActionCont must be univalent. □

The extension of an action container, $[\![S \triangleleft P \triangleright^{\sigma} G]\!](X) \coloneqq \sum_{s:S}(P_s \to X) \,/\, \sigma_s$, is again functorial:

Proposition 3.2.13. The extension of an action containers defines a full functor $[\![\_]\!] : \mathsf{ActionCont} \to \mathsf{Endo}(\mathsf{hSet})$. □

Unlike $[\![\_]\!]_/$ for quotient containers, this functor is not faithful; morphisms of action containers whose position-maps differ by a permutation may map to the same natural transformation.[2]

We show that action containers admit all products and coproducts by lifting properties of Action through Fam. First, recall that a direct product of groups acts on sums of sets. This can be expressed nicely as an action of a Π-type on a Σ-type:

Problem 3.2.14. Given a set S and a families of sets $P : S \to \mathsf{hSet}$, groups $S : G \to \mathsf{Group}$ and actions $\sigma : \prod_{s:S} G_s \dot{\to} \Sigma(P_s)$, define an action of the direct product $\prod_{s:S} G_s$ on $\sum_{s:S} P_s$.

[2] The relation $\approx$ on premorphisms (definition 3.1.6) exactly measures this failure of being faithful.

Construction. Given a section $\gamma : \prod_{s:S} G_s$, we need to define some $\sum_S P \simeq \sum_S P$. We do so by applying $\sigma_s(\gamma(s)) : P_s \simeq P_s$ to the second projection of the Σ-types. It is straightforward to show that this defines a group homomorphism. ⌟

This canonical action defines the cartesian products of Action:

Proposition 3.2.15. Action has S-fold products for all sets S. When S is empty, this implies that the trivial group acting on the empty set is a terminal object.

Proof. The base category $\mathsf{Group} \times \mathsf{hSet}^{\mathsf{op}}$ is closed under S-fold products; on objects, the products are direct products of groups, $\prod_{s:S} G_s$, and *sums* of sets, $\sum_{s:S} P_s$. For each family $\sigma : \prod_{s:S} \mathsf{Action}_0(G_s, P_s)$, it suffices to give a displayed product, namely some $\mathsf{Action}_0(\Pi G, \Sigma P)$. But problem 3.2.14 gives us exactly that: an action of direct products on sums. It remains to show that the projection maps $\pi_s : (\Pi G, \Sigma P) \to (G_s, P_s)$ are equivariant, and that they are suitably universal. The first holds by definition, the second follows easily as isEquivariant is a proposition.

If S is empty, then $\prod_S G \simeq 1$ and $\sum_S P \simeq 0$, and the induced action is a map of trivial groups $1 \xrightarrow{\cdot} \Sigma(0)$. □

Products and coproducts of families have concise type-theoretic formulations. For any $\mathcal{C}$, the category $\mathsf{Fam}(\mathcal{C})$ defines its *free coproduct completion*, so coproducts of families are simply dependent sums of families:

Proposition 3.2.16. $\mathsf{Fam}(\mathcal{C})$ has all coproducts.

Proof. Let $K : \mathsf{hSet}$ and $x : K \to \mathsf{Fam}(\mathcal{C})_0$. At each $k : K$, x_k consists of an indexing set $I_k : \mathsf{hSet}$ and a family of objects $X_k : I_k \to \mathcal{C}_0$. The K-fold coproduct of x is a family $\Sigma(x) : \mathsf{Fam}(\mathcal{C})_0$, indexed by $\sum_{k:K} I_k$, with elements

$$\begin{aligned} \textstyle\sum_{k:K} I_k &\to \mathcal{C}_0 \\ (k, i) &\mapsto X_k(i) \end{aligned}$$

By currying, morphisms out of $\Sigma(x)$ factor uniquely through the obvious injections $\iota_k : \mathsf{Fam}(\mathcal{C})_1(x_k, \Sigma(x))$. □

That $\mathsf{Fam}(\mathcal{C})$ freely adds coproducts to $\mathcal{C}$ was known to Bénabou [Bén85, (3.5)]. Adámek and Rosický give a comprehensive overview of many other "folklore" properties of $\mathsf{Fam}(\mathcal{C})$ in [AR20, §2]. We show that Fam preserves products by adapting the slightly more general [AR20, Lemma 2.6]:

Proposition 3.2.17. If $\mathcal{C}$ has all products, then so does $\mathsf{Fam}(\mathcal{C})$.

Proof. Let K and $x_k \doteq (I_k, X_k)$ as in proposition 3.2.16. The K-fold product of x is a family $\Pi(x) : \mathsf{Fam}(\mathcal{C})_0$, this time indexed by sections $\prod_{k:K} I_k$, with elements

$$\begin{aligned} \textstyle\prod_{k:K} I_k &\to \mathcal{C}_0 \\ \varphi &\mapsto \textstyle\prod^{\mathcal{C}}_{k:K} X_k(\varphi(k)) \end{aligned}$$

The projections $\pi_k : \mathsf{Fam}(\mathcal{C})_1(\Pi(x), x_k)$ are given pointwise by the projections in $\mathcal{C}$. Morphisms into $\Pi(x)$ factor uniquely through π_k since they do so pointwise in $\mathcal{C}$. □

Proposition 3.2.18. Action containers admit all products and coproducts.

Proof. ActionCont has all coproducts "for free" (proposition 3.2.16), and Fam preserves (proposition 3.2.17) the products of Action (proposition 3.2.15). □

Constant datatypes are representable as action containers: their sets of positions are empty, hence the unit group 1 acts trivially on their singular symmetry $0 \simeq 0$.

Definition 3.2.19. The *constant container* of a set $K : \mathsf{hSet}$ is $\mathbf{k}\,K \coloneqq (K \triangleleft 0 \triangleright^{\tau} 1)$, where $\tau : 1 \xrightarrow{\cdot} \Sigma(0)$ is the trivial action. ⌟

Constants are exponentiable in the category of action containers, just like they are in the category of containers [AAG05, Proposition 2.8]:

Proposition 3.2.20. Given $F \doteq (S \triangleleft P \triangleright^{\sigma} G)$, and $K : \mathsf{hSet}$, the K-fold product container $F^K \coloneqq \prod_{k:K} F$ is an exponential object with evaluation map

$$\mathsf{ev} : F^K \otimes \mathbf{k}\,K \to F$$

given by the following data:

- on shapes, by function application $\mathsf{ev}_{\mathsf{Sh}} : (K \to S) \times K \to S$,
- on positions, by pairing $\mathsf{ev}_{\mathsf{Ps}}(f,k) : P_{fk} \to 0 + \sum_{k':K} P_{fk'}$, and
- on symmetries, by the projections $\mathsf{ev}_{\mathsf{Sm}}(f,k) : 1 \times \prod_{k':K} G_{fk'} \xrightarrow{\cdot} G_{fk}$

3.3 Embedding action containers in symmetric containers

In § 3.1.4, we observed that quotient containers lift to symmetric containers. Unfortunately, their morphisms do not properly preserve symmetries, hence the lifting fails to be functorial. We defined the category of action containers to include the missing data, and we are now ready to define an appropriate lifting. Symmetric containers do not form an ordinary category, but a 2-category instead:

Proposition 3.3.1. Symmetric containers, their morphisms and identification of morphisms form a 2-category SymmCont.

Proof. $(1,0)$-truncated containers form a wild category whose types of morphisms are h-groupoids. By lemma 1.5.17 this induces the structure of a 2-category in which 2-cells are identifications of morphisms. □

Thus, in order to show that delooping of containers is functorial, we must first equip action containers with a type of 2-cells, defining a 2-category. Instead of plucking 2-cells out of thin air, we pull back 2-cells of symmetric containers (i.e. homotopies of container morphisms), and show that these cells closely resemble the quotients of premorphisms of quotient containers (definition 3.1.6). The goal of this section is to show that delooping of action containers embeds them in symmetric containers in a suitable, 2-categorical sense.

From a homotopical perspective, a symmetric container is nothing but a *set bundle* over an *h-groupoid*: the container $(B \triangleleft F)$ consist of some *base* $B : \mathsf{hGrpd}$ over which a family of *fibers* $F : B \to \mathsf{hSet}$ lives. In order to show that delooping of action containers is functorial, we split its construction into smaller steps. As in § 3.2.1, before considering families, we first seek to understand the problem for a single action σ, and in particular its associated *bundle* $\bar{\mathbf{B}}\sigma : \mathbf{B}G \to \mathsf{hSet}$.

We define 2-categories of actions (problem 3.3.13) and set bundles (problem 3.3.14), and show that taking an action to its associated bundle (definition 3.1.18) is a pseudofunctor. We define the 2-cells of actions in such a way that they express, in the language of groups, exactly the data of homotopies of bundle maps. Formally, we prove that the pseudofunctor $\bar{\mathbf{B}}$ is locally a weak equivalence (theorem 3.3.20), i.e. an embedding of 2-categories. This embedding tells us that morphisms of single-shape action containers represent exactly morphisms of single-shape symmetric containers, and that homotopies of the latter have an explicit description in terms of 2-cells of action containers.

To understand delooping for many-shape containers, we give a 2-categorical Fam-construction and let ActionCont := Fam(Action). We ensure that objects and morphisms of this construction coincide with their 1-categorical analogues. Unfolding the type of 2-cells introduced by this construction, we show that it is closely related to the quotient on premorphisms of quotient containers (proposition 3.3.25).

Any family of set bundles can be seen as a single bundle (over the sum of their bases), and this defines a pseudofunctor $\Sigma : \mathsf{Fam}(\mathsf{Bundle}) \to \mathsf{Bundle}$. The

delooping of action containers then factors into

$$\begin{array}{ccccc} \mathsf{ActionCont} & & \dashrightarrow & & \mathsf{SymmCont} \\ {\scriptstyle =}\downarrow & & & & \uparrow{\scriptstyle =} \\ \mathsf{Fam}(\mathsf{Action}) & \xrightarrow[\mathsf{Fam}(\bar{\mathbf{B}})]{} & \mathsf{Fam}(\mathsf{Bundle}) & \xrightarrow[\Sigma]{} & \mathsf{Bundle} \end{array} \tag{3.2}$$

We overload notation and write $\mathbf{B} : \mathsf{ActionCont} \to \mathsf{SymmCont}$ for the resulting 2-functor. This factorization lets us prove, step by step, that the interpretation of action containers as symmetric containers is sound: delooping is locally fully-faithful (theorem 3.3.31), hence morphisms of action containers embed into those of symmetric containers.

As a warmup, let us define the action of $\mathbf{B}$ on action containers and their morphisms by hand.

Definition 3.3.2. The *delooping* of an action container $(S \triangleleft P \triangleright^{\sigma} G)$ is the symmetric container

$$\mathbf{B}(S \triangleleft P \triangleright^{\sigma} G) \coloneqq \left((s,x) : \textstyle\sum_{s:S} \mathbf{B}G_s \triangleleft \bar{\mathbf{B}}\sigma_s(x)\right) \qquad \lrcorner$$

The above is the same as for quotient containers. In particular, the extent of an action container and its delooping agree:

Proposition 3.3.3 (analogue of theorem 3.1.25)**.** For all $F : \mathsf{ActionCont}$, the functors $\|[\![\mathbf{B}F]\!](\_)\|_0$ and $[\![F]\!]$ are naturally isomorphic, hence propositionally equal.

We failed to lift morphisms of quotient containers (cf. equation (3.1)), but we can now implement the following for morphisms of action containers:

Problem 3.3.4. Given a morphism of action containers $F \to G$, define a morphism between their deloopings, $\mathbf{B}F \to \mathbf{B}G$.

Construction. Let $F \doteq (S \triangleleft P \triangleright^{\sigma} G)$, $G \doteq (T \triangleleft Q \triangleright^{\tau} H)$, and $(u \triangleleft f \triangleright \varphi) : F \to G$. The group homomorphisms φ lift (problem 3.1.17) to a map of shapes

$$\begin{aligned} &\bar{\varphi} : \textstyle\sum_{s:S} \mathbf{B}G_s \to \sum_{t:T} \mathbf{B}H_t \\ &\bar{\varphi}(s,x) \coloneqq (u(s), \mathbf{B}\varphi_s(x)) \end{aligned}$$

The map on positions has (uncurried) type

$$\bar{f} : \prod_{s:S} \prod_{x:\mathbf{B}G_s} \bar{\mathbf{B}}\tau_{us}(\mathbf{B}\varphi_s(x)) \to \bar{\mathbf{B}}\sigma_s(x)$$

On the point, this type reduces to $\bar{f}(s,\bullet) : Q_{us} \to P_s$, and we set $\bar{f}(s,\bullet) \coloneqq f_s$. It remains to show that $\bar{f}$ is well-defined on loops in $\mathbf{B}G_s$. For all $g : G_s$, we have to provide a dependent path $f_s =^{Y}_{\mathsf{loop}\, g} f_s$, where $Y : \mathbf{B}G_s \to \mathsf{Type}$ is the family

$$Y(x) \coloneqq \bar{\mathbf{B}}\tau_{us}(\mathbf{B}\varphi_s(x)) \to \bar{\mathbf{B}}\sigma_s(x)$$

By Lemma 2.9.6 of [Uni13] and function extensionality, this is equivalent to giving a homotopy

$$\begin{array}{ccc} P_s & \xleftarrow{f_s} & Q_{us} \\ {\scriptstyle \sigma_s(g)}\downarrow & & \downarrow{\scriptstyle \tau_{us}(\varphi_s(g))} \\ P_s & \xleftarrow[f_s]{} & Q_{us} \end{array}$$

But we assume f_s to be an equivariant map, whence the diagram commutes. ⌟

3.3.1 Delooping, 2-categorically

The core of our construction is the fact that a group is nothing but a single-object groupoid. Despite being blatantly obvious, this observation equips groups with the structure of a 2-category: The prototypical 2-category is that of categories, functors and natural transformations. In its full subcategory of (categorical) groupoids, all natural transformations are necessarily isomorphisms. Further restricting the objects to single-object groupoids, functors are nothing but group homomorphisms. It is in this 2-category that natural isomorphisms define a canonical notion of 2-cell between group homomorphisms. In this section, we give an explicit description of these 2-cells in purely group-theoretic terms, and show that it embeds, as a 2-category, into groupoids. However, instead of defining a 2-category of categorical groupoids, we take the univalent perspective:[3]

Definition 3.3.5. The $(2,1)$-category hGrpd has as objects h-groupoids. Morphisms are plain maps, and (invertible) 2-cells are given by their homotopies. ⌟

We show that delooping extends to a pseudofunctor $\mathbf{B} : \mathsf{Group} \to \mathsf{hGrpd}$ which is locally an equivalence of categories, hence defines the desired embedding. In other applications, this 2-categorical perspective on groups is useful to understand, for example, wreath products [Yua14] or inner automorphisms of groups [HK24]. Here, we would like to give a type-theoretic account, only depending on the higher inductive type $\mathbf{B}G$ and its associated induction principle.

First, observe that each identification of group homomorphisms under $\mathbf{B}$ can be expressed by a single group element:

[3] It is not hard to see that, by univalence, both are propositionally *equal* 2-categories.

Problem 3.3.6. Let $G, H : \mathsf{Group}$ and $\varphi, \psi : G \dashrightarrow H$. Construct an equivalence of types

$$(\mathbf{B}\varphi = \mathbf{B}\psi) \simeq \sum_{h:H} \prod_{g:G} \varphi(g) \cdot h = h \cdot \psi(g)$$

Construction. By function extensionality, we have

$$(\mathbf{B}\varphi = \mathbf{B}\psi) \simeq \prod_{x:\mathbf{B}G} \mathbf{B}\varphi(x) = \mathbf{B}\psi(x)$$

On the right, $P(x) :\equiv \mathbf{B}\varphi(x) = \mathbf{B}\psi(x)$ is a family of sets over $\mathbf{B}G$, so by proposition 3.1.14.(1) we can characterize the product as

$$\simeq \sum_{p_0 : P(\bullet)} \prod_{g:G} p_0 =^P_{\mathsf{loop}(g)} p_0$$

Since $P(\bullet) = (\bullet = \bullet)$, the type $p_0 =^P_{\mathsf{loop}(g)} p_0$ is expressible in terms of composite paths,

$$\simeq \sum_{p_0 : \bullet = \bullet} \prod_{g:G} \mathsf{loop}(\varphi(g)) \cdot p_0 = p_0 \cdot \mathsf{loop}(\psi(g))$$

By lemma 3.1.16, $\mathsf{loop} : H \to (\bullet =_{\mathbf{B}H} \bullet)$ is an equivalence, hence

$$\simeq \sum_{h:H} \prod_{g:G} \mathsf{loop}(\varphi(g)) \cdot \mathsf{loop}(h) = \mathsf{loop}(h) \cdot \mathsf{loop}(\psi(g))$$

Morover, it preserves the multiplication of H, i.e.

$$\simeq \sum_{h:H} \prod_{g:G} \mathsf{loop}(\varphi(g) \cdot_H h) = \mathsf{loop}(h \cdot_H \psi(g))$$

Cancelling loop once more, we obtain the desired type

$$\simeq \sum_{h:H} \prod_{g:G} \varphi(g) \cdot_H h = h \cdot_H \psi(g)$$ ⌟

Notice how this type resembles a set of natural transformations: each h is the (unique) component of a transformation from φ to ψ; it is natural with respect to the functorial actions $\varphi(g)$ and $\psi(g)$ on a "morphism" g. Intuitively, $\mathbf{B}$ identifies homomorphisms if they differ by exactly a conjugation, hence we give the following name:

Definition 3.3.7. Let G, H : Group. We say that $r : H$ is a *conjugator* of two homomorphisms $\varphi, \psi : G \dashrightarrow H$ if it identifies them up to conjugation:

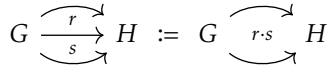
$$\mathsf{isConjugator}_{\varphi,\psi} : H \to \mathsf{Prop}$$
$$\mathsf{isConjugator}_{\varphi,\psi}(r) \coloneqq \prod_{g:G} \varphi(g) \cdot r = r \cdot \psi(g)$$

We denote the type of all conjugators by $\mathsf{Conjugator}(\varphi, \psi) \coloneqq \sum_{r:H}(\varphi \approx_r \psi)$. We abbreviate $\varphi \approx_r \psi \coloneqq \mathsf{isConjugator}_{\varphi,\psi}(r)$ and $\varphi \approx \psi \coloneqq \mathsf{Conjugator}(\varphi, \psi)$, unless ambiguous. ⌟

As desired, conjugators form invertible 2-cells in a 2-category of groups:

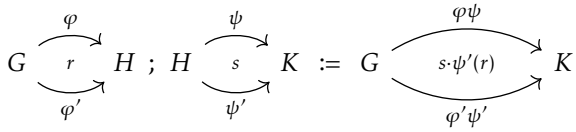
Problem 3.3.8. Assemble groups, homomorphisms, and conjugators into a $(2,1)$-category Group.

Construction. Morphisms compose as usual, so it remains to give composition of 2-cells. Vertically, conjugators compose by multiplication:



Indeed, if $\varphi \approx_r \psi$ and $\psi \approx_s \rho$, then $\varphi(g)rs = r\psi(g)s = rs\rho(g)$ for all $g : G$, hence $\varphi \approx_{rs} \rho$. The identity 2-cell on any $\varphi : G \dashrightarrow H$ is given by the unit of H: clearly, $\varphi \approx_{1_H} \varphi$. Since composition is just multiplication in H, it is unital and associative. This defines the local structure of a category on morphisms. All 2-cells are invertible, since $\varphi \approx_r \psi$ implies $\psi \approx_{r^{-1}} \varphi$.

Horizontal composition is obtained similarly:



That $s \cdot \psi'(r)$ is indeed a conjugator (i.e. $\varphi\psi \approx_{s\psi'(r)} \varphi'\psi'$) depends on both r and s being conjugators.[4] Verifying that this structure satisfies the axioms of a 2-category is entirely straightforward. ⌟

Problem 3.3.9. Extend the delooping of groups to a 2-functor $\mathbf{B}$: Group $\to$ hGrpd.

[4] In particular $\psi(r) \cdot s = s \cdot \psi'(r)$, so which one we pick as the composite does not matter. We choose the right option so that it resembles the order of composition for 1-cells.

Construction. On groups and homomorphisms, $\mathbf{B}$ acts as given by definition 3.3.2 and problem 3.3.4, respectively. The inverse map underlying the equivalence of problem 3.3.6 gives an action on 2-cells, i.e. a map $\mathbf{B}_2 : \varphi \approx \psi \to \mathbf{B}(\varphi) = \mathbf{B}(\psi)$. That this map appropriately preserves horizontal and vertical composition is easy to prove by function extensionality and $\mathbf{B}$-induction. Consider, for example $\varphi : G \dot{\to} H$. To show that $1_H : \varphi \approx \varphi$ maps to $\mathsf{refl} : \mathbf{B}\varphi = \mathbf{B}\varphi$, it suffices to show $\forall(x : \mathbf{B}G).\, \mathbf{B}_2(1_H)_*(x) = \mathsf{refl}$. But this is a proposition, hence it is enough to establish it at $x \doteq \bullet$. Unfolding the definition of $\mathbf{B}_2$, this amounts to $\mathsf{loop}(1_H) = \mathsf{refl}$, which holds by lemma 3.1.16. ⌟

By construction, $\mathbf{B}$ embeds conjugators as 2-cells of h-groupoids — that is, homotopies of maps:

Proposition 3.3.10. Delooping is locally fully-faithful: For groups G, H, the local functor $\mathsf{Group}_1(G,H) \to \mathsf{hGrpd}_1(\mathbf{B}G, \mathbf{B}H)$ is fully-faithful.

Proof. We need to show that, for any pair of morphisms $\varphi, \psi : \mathsf{Group}_1(G,H)$, the action on 2-cells $\mathsf{Group}_2(\varphi,\psi) \to \mathsf{hGrpd}_2(\mathbf{B}\varphi, \mathbf{B}\psi)$ is an equivalence of sets. But by definition, this action is the equivalence of problem 3.3.6. □

Furthermore, it fully reflects the local structure of hGrpd: Every map of groupoids $\mathbf{B}G \to \mathbf{B}H$ (merely) arises from a group homomorphism.

Proposition 3.3.11. Delooping is locally essentially surjective.

Proof. Let $G, H : \mathsf{Group}$ and $f : \mathbf{B}G \to \mathbf{B}H$ a map of h-groupoids. We show that there *merely* exists some $\varphi : \mathsf{Group}_1(G,H)$ together with an isomorphism $\mathbf{B}\varphi \cong f$ in the local category $\mathsf{hGrpd}(\mathbf{B}G, \mathbf{B}H)$. By definition, isomorphisms in the latter category are homotopies of maps, hence it suffices to exhibit some $h : \prod_{x:\mathbf{B}G} \mathbf{B}\varphi(x) = f(x)$. Since $\mathbf{B}H$ is a *connected* groupoid, there *merely* exists a path $p : f(\bullet) = \bullet$. Conjugation by p is a map

$$\begin{aligned} &\mathsf{conj}(p) : f(\bullet) =_{\mathbf{B}H} f(\bullet) \to \bullet =_{\mathbf{B}H} \bullet \\ &\mathsf{conj}(p) \coloneqq \lambda q.\, p^{-1} \cdot q \cdot p \end{aligned}$$

This is in particular a homomorphism of groups, $\mathsf{conj}(p) : \Omega(\mathbf{B}H, f(\bullet)) \dot{\to} \Omega(\mathbf{B}H, \bullet)$. We have a second homomorphism induced by the action of f on paths, namely $\mathsf{cong}(f) : \Omega(\mathbf{B}G, \bullet) \dot{\to} \Omega(\mathbf{B}H, f(\bullet))$. Putting these together with the equivalence of groups loop, we obtain φ as the composite homomorphism

$$\begin{array}{ccccc} G & & \xrightarrow{\qquad\qquad\varphi\qquad\qquad} & & H \\ {\scriptstyle \mathsf{loop}}\downarrow & & & & \uparrow{\scriptstyle \mathsf{loop}^{-1}} \\ \bullet =_{\mathbf{B}G} \bullet & \xrightarrow[\mathsf{cong}(f)]{} & f(\bullet) =_{\mathbf{B}H} f(\bullet) & \xrightarrow[\mathsf{conj}(p)]{} & \bullet =_{\mathbf{B}H} \bullet \end{array}$$

We give h by induction on $\mathbf{B}G$. On the point, $h(\bullet) : \bullet = f(\bullet)$ is given by p^{-1}. On loops, we construct $\prod_{g:G} \mathbf{B}\varphi(\mathsf{loop}\, g) \cdot p^{-1} = p^{-1} \cdot f(\mathsf{loop}\, g)$ as follows: let $g : G$, then

$$\mathbf{B}\varphi(\mathsf{loop}\, g) \cdot p^{-1} \doteq \mathsf{loop}(\mathsf{loop}^{-1}(p^{-1} \cdot f(\mathsf{loop}\, g) \cdot p)) \cdot p^{-1} = p^{-1} \cdot f(\mathsf{loop}\, g) \qquad \square$$

Together, these two properties fully characterize 2-cells of Group: Delooping is an embedding of 2-categories.

Theorem 3.3.12. $\mathbf{B} : \mathsf{Group} \to \mathsf{hGrpd}$ is locally a *weak* equivalence of categories. In particular, there merely exist inverse functors $\mathsf{hGrpd}(\mathbf{B}G, \mathbf{B}H) \to \mathsf{Group}(G, H)$ for all $G, H : \mathsf{Group}$.

Proof. By definition 1.5.9, fully-faithful and essentially surjective functors are weak equivalences. $\square$

Note that this cannot be strengthened to a full equivalence with explicit inverse: equivalence of categories preserves properties, yet hGrpd is by definition locally univalent, whereas Group is not: the identity type of homomorphisms $\varphi = \psi$ is a proposition, but $\varphi \approx \psi$ is in general a non-trivial set. In § 3.4, we equip hGrpd with extra structure such that groups embed into such structured groupoids *as a 1-category*.

3.3.2 A 2-category of actions

Stretching the analogy of groups as single-object objects further, it is not hard to see that any G-action on a set X is equivalently a functor from G into sets — its value on the single object is X, and each morphism is some $g : G$, which is mapped to a permutation of X. Internally, we have already implemented this as $\bar{\mathbf{B}} : \mathbf{B}G \to \mathsf{hSet}$, the associated bundle. We will now define a pair of 2-categories — actions and set-bundles — and show that $\bar{\mathbf{B}}$ is a functorial construction, taking one to the other.

Problem 3.3.13 (2-category of group actions)**.** Show that the following data define a *locally thin* 2-category displayed over Group:

- For each $G : \mathsf{Group}_0$, objects are G-actions:

$$\mathsf{Action}_0(G) :\equiv \sum_{X:\mathsf{hSet}} (G \mathrel{\dot{\to}} \Sigma X)$$

- Given actions $(X, \sigma) : \mathsf{Action}_0(G)$ and $(Y, \tau) : \mathsf{Action}_0(H)$, the 1-cells over $\varphi : \mathsf{Group}_1(G, H)$ are equivariant maps:

$$\mathsf{Action}_1(\varphi; (X, \sigma), (Y, \tau)) :\equiv \sum_{f:X \leftarrow Y} \mathsf{isEquivariant}_{\sigma,\tau}(\varphi, f)$$

- Given equivariant maps $f : (X, \sigma) \to_\varphi (Y, \tau)$ and $g : \psi, (X, \sigma) \to_\psi (Y, \tau)$, the 2-cells over a conjugator $r : \mathsf{Group}_2(\varphi, \psi)$ are identifications of f and g up to permutation by r:

$$\begin{aligned}&\mathsf{Action}_2(r; f, g) : \mathsf{Prop}\\ &\mathsf{Action}_2(r; f, g) \coloneqq (f = g \circ \tau(r))\end{aligned}$$

Denote by Action its total 2-category.

Construction. Equivariant maps compose just like they do in problem 3.2.7. The type of 2-cells is a proposition, hence verifying the axioms of a displayed 2-category reduces to defining *some* identity and composite 2-cells. The vertical composite $f \Rightarrow_{r \cdot s} h$ of 2-cells $p : f \Rightarrow_r g$ and $q : g \Rightarrow_s h$ is some identification $f = g \circ \tau(rs)$. Here, τ is the action on the domain of f, and we derive

$$f \overset{p}{=} g \circ \tau(r) \overset{q}{=} h \circ \tau(s) \circ \tau(r) = h \circ \tau(rs)$$

Horizontal composition is defined similarly, and depends on 2-cells of Group being conjugators. ⌟

A *set bundle* consists of some $B : \mathsf{Type}$ (the *base*), over which there is a family $F : B \to \mathsf{hSet}$ of *fibers*. Given a point $b : B$, we think of $F(b)$ as a space of (topologically) discrete points that lie over b in the base. In the universal cover of the circle, $\mathfrak{c} : S^1 \to \mathsf{Type}$ [Uni13, § 8.1.3], for example, the fibers over $\mathsf{base} : S^1$ are the integers, $\mathfrak{c}(\mathsf{base}) \doteq \mathbb{Z}$. The associated bundle of an action is, of course, a set bundle whose base is an h-groupoid. Set bundles over h-groupoids assemble into a 2-category:

Problem 3.3.14 (2-category of set bundles)**.** Define a *locally thin* 2-category displayed over hGrpd with the following data:

- Given $G : \mathsf{hGrpd}_0$, set bundles on G are families

$$\mathsf{Bundle}_0(G) \coloneqq (G \to \mathsf{hSet})$$

- Over $\varphi : \mathsf{hGrpd}_1(G, H)$, morphisms from $X : \mathsf{Bundle}_0(G)$ to $Y : \mathsf{Bundle}_0(H)$ are families of maps

$$\mathsf{Bundle}_1(\varphi; X, Y) \coloneqq \prod_{g:G} Y(\varphi g) \to X(g)$$

- Over $p : \varphi = \psi$, displayed 2-cells from $f : X \to_\varphi Y$ to $g : X \to_\psi Y$ are dependent identifications

$$\mathsf{Bundle}_2(p; f, g) : \mathsf{Prop}$$
$$\mathsf{Bundle}_2(p; f, g) :\equiv f =_p^B g$$

  where $B(\rho) :\equiv \prod_{g:G} Y(\rho g) \to X(g)$.

Construction. For 1-cells, composition of families of maps is just a family of composite maps. Displayed 2-cells are dependent paths, hence canonically compose horizontally as well as vertically. Since the type of 2-cells is a proposition, composition is automatically coherent. ⌟

Remark 3.3.15. Every symmetric container $(S \triangleleft P)$ is, by definition, a set bundle. In fact, the above construction of a 2-category of bundles is exactly that of the 2-category of symmetric containers in proposition 3.3.1.

In the following we will use the identification $\mathsf{SymmCont} \doteq \mathsf{Bundle}$ to freely change our perspective on problems. ⌟

We can lift equivariant maps to maps between associated bundles as expected:

Problem 3.3.16. Let σ and τ be actions of G on X and H on Y, respectively. If $f : X \leftarrow Y$ is an equivariant map with respect to some $\varphi : G \dot{\to} H$, lift it to a family of maps

$$\bar{\mathbf{B}} f : \prod_{x:\mathbf{B}G} \bar{\mathbf{B}}\tau(\mathbf{B}\varphi(x)) \to \bar{\mathbf{B}}\sigma(x)$$

between the associated bundles $\bar{\mathbf{B}}\sigma$ and $\bar{\mathbf{B}}\tau$.

Construction. The map is defined by induction. On the point, $\bar{\mathbf{B}} f(\bullet) : Y \to X$ is given by f itself. On a loop, we need to give a path of type

$$\bar{\mathbf{B}}\sigma(\mathsf{loop}\, g) \circ f = f \circ \bar{\mathbf{B}}\tau(\mathbf{B}\varphi(\mathsf{loop}\, g)),$$

but this type reduces to $\sigma(g) \circ f = f \circ \tau(\varphi(g))$, which is inhabited because we assume f to be equivariant. ⌟

This lifting respects identifications of equivariant maps up to a conjugator: if parallel equivariant maps are related by a conjugator, then there exists a homotopy between their associated maps of bundles.

Lemma 3.3.17. Let $\sigma : G \dot{\to} \Sigma(X)$ and $\tau : H \dot{\to} \Sigma(Y)$. Assume that $f, g : Y \to X$ are equivariant maps (in the sense of definition 3.2.5) over some $\varphi, \psi : G \dot{\to} H$, respectively. If $r : \varphi \approx \psi$ and $f = g \circ \tau(r)$, then there exists a homotopy $\bar{\mathbf{B}} f =_p \bar{\mathbf{B}} g$ over the path $p :\equiv \mathbf{B}(r) : \mathbf{B}\varphi = \mathbf{B}\psi$.

Proof. It suffices to construct the homotopy $\bar{\mathbf{B}}f =_p \bar{\mathbf{B}}g$ pointwise, that is to give some $h : \prod_{x:\mathbf{B}G} \bar{\mathbf{B}}f(x) =_p \bar{\mathbf{B}}g(x)$. This is a proposition, hence it is fully determined by $h(\bullet)$, whose type reduces to $f =^{B}_{\mathsf{ua}(\tau(r))} g$ where $B(Z : \mathsf{hSet}) \coloneqq Z \to X$. But by univalence, this is equivalent to the assumption that $f = g \circ \tau(g)$. □

Both Action and Bundle are defined as total categories, and $\mathbf{B} : \mathsf{Group} \to \mathsf{hGrpd}$ is a 2-functor between their bases. Thus, in order to define the 2-functor $\bar{\mathbf{B}} : \mathsf{Action} \to \mathsf{Bundle}$, it suffices to specify it over the displayed structure. The previous results provide exactly these missing data.

Proposition 3.3.18. Taking associated bundles extends to a displayed 2-functor $\bar{\mathbf{B}} : \mathsf{Action} \to_{\mathbf{B}} \mathsf{Bundle}$, defined by the following data:

- On objects, it sends an action to its associated bundle, $\bar{\mathbf{B}}(G;(X,\sigma)) \coloneqq \bar{\mathbf{B}}\sigma$.
- On morphisms, it associates to an equivariant map $f : \mathsf{Action}_1(\varphi;\sigma,\tau)$ a bundle map $\bar{\mathbf{B}}f : \mathsf{Bundle}_1(\mathbf{B}\varphi, \bar{\mathbf{B}}\sigma, \bar{\mathbf{B}}\tau)$, as given by problem 3.3.16.
- Over $r : \mathsf{Group}_2(\varphi,\psi)$, a displayed 2-cells $p : \mathsf{Action}_2(r; f, g)$ is sent to the homotopy of bundle maps of lemma 3.3.17.

Proof. A simple proof by induction ensures that $\bar{\mathbf{B}}$ preserves 1-dimensional identities and composition. Since 2-cells are propositions it trivially preserves the 2-dimensional structure. □

Definition 3.3.19. $\bar{\mathbf{B}} : \mathsf{Action} \to \mathsf{Bundle}$ is the total 2-functor of the displayed functor given in proposition 3.3.18. ⌟

We would like to remark that splitting the construction of $\bar{\mathbf{B}}$ into a "base" and "displayed" part neatly separates concerns: The hard work is done ensuring that on bases, $\mathbf{B} : \mathsf{Group} \to \mathsf{hGrpd}$ properly preserves the 2-dimensional structure. On the displayed part, the 2-dimensional behaviour of $\bar{\mathbf{B}}$ is automatically coherent by virtue of Bundle having propositional displayed 2-cells. Ensuring directly that the total functor satisfies all the necessary coherences is, in comparison, unwieldy, despite amounting to the same proof obligations. This will become even more apparent in the proof of the following embedding theorem:

Theorem 3.3.20. $\bar{\mathbf{B}} : \mathsf{Action} \to \mathsf{Bundle}$ is locally a weak equivalence.

Proof. $\bar{\mathbf{B}}$ is locally fully-faithful if its action on 2-cells is an equivalence. By virtue of $\bar{\mathbf{B}}$ being a total functor, this is a map on Σ-types, with $\mathbf{B}_2$ and $\bar{\mathbf{B}}_2$ acting on the first and second projection, respectively. $\mathbf{B}_2$ is an equivalence by proposition 3.3.10. In lemma 3.3.17, $\bar{\mathbf{B}}_2$ is defined using function extensionality

and $\mathbf{B}$-induction, both of which induce equivalences, hence it is an equivalence $\mathsf{Action}_2(r; f, g) \simeq \mathsf{Bundle}_2(\mathbf{B}_2(r); \bar{\mathbf{B}}_1(f), \bar{\mathbf{B}}_1(g))$ for appropriately typed r, f, g.

Local essential surjectivity follows similarly. Given $\sigma, \tau : \mathsf{Action}_0$ and a bundle map $(\varphi, f) : \mathsf{Bundle}_1(\bar{\mathbf{B}}_0(\sigma), \bar{\mathbf{B}}_0(\tau))$, we have to merely exhibit some $(\varphi^*, f^*) : \mathsf{Action}_1(\sigma, \tau)$ such that $\bar{\mathbf{B}}_1(\varphi^*, f^*) \cong (\varphi, f)$. An isomorphism in a total category is exactly a pair of an isomorphism in the base and a displayed isomorphism over it, i.e. our goal is to give $p : \mathbf{B}_1(\varphi^*) \cong \varphi$ and $\bar{\mathbf{B}}_1(f^*) \cong_p f$. Local essential surjectivity of $\mathbf{B}$ (proposition 3.3.11) (merely) gives us φ^* and p. By inspecting the construction in problem 3.3.16, we see that action on morphisms is an equivalence

$$\bar{\mathbf{B}}_1 : \mathsf{Action}_1(\varphi; \sigma, \tau) \simeq \mathsf{Bundle}_1(\mathbf{B}_1\varphi; \bar{\mathbf{B}}_0(\sigma), \bar{\mathbf{B}}_0(\tau))$$

since it is defined by induction on $\mathbf{B}$. Hence we can define $f^* \coloneqq \bar{\mathbf{B}}_1^{-1}(f)$, up to some transport along p. □

Notice that in the above proof, most of the time is spent spelling out the proof obligations. The actual proofs follow directly by inspecting the actions of displayed functor $\bar{\mathbf{B}}$ on 1- and 2-cells.

3.3.3 Action containers as a 2-category of families

In the previous section we have seen how containers with a single shape relate to set bundles. Since our goal is to define a 2-category of action containers in terms of families of actions, we introduce a 2-categorical Fam-construction, again employing displayed machinery.

Definition 3.3.21 (2-category of families)**.** Let $\mathcal{C}$ be a 2-category. The 2-category $\mathsf{Fam}(\mathcal{C})$ displayed over hSet consists of the following data:

- Over $J : \mathsf{hSet}_0$, displayed objects are families of $\mathcal{C}$:

$$\mathsf{Fam}_0(\mathcal{C}) \coloneqq J \to \mathcal{C}_0$$

- Given families $X : \mathsf{Fam}_0(J)$ and $Y : \mathsf{Fam}_0(K)$, the type of 1-cells displayed over some $\varphi : \mathsf{hSet}_1(J, K)$ is

$$\mathsf{Fam}_1(\varphi; X, Y) \coloneqq \prod_{j:J} \mathcal{C}_1(X_j, Y_{\varphi j})$$

- Given $\varphi, \psi : \mathsf{hSet}_1(J, K)$, displayed 2-cells are defined by path induction on homotopies $\varphi = \psi$:

$$\mathsf{Fam}_2 : (\varphi = \psi) \to \mathsf{Fam}_1(\varphi; X, Y) \to \mathsf{Fam}_1(\psi; X, Y) \to \mathsf{hSet}$$
$$\mathsf{Fam}_2(\mathsf{refl}_\varphi; f, g) :\equiv \prod_{j:J} \mathcal{C}_2(f_j, g_j)$$

⌟

As usual, we denote by $\mathsf{Fam}(\mathcal{C})$ both the displayed 2-category and its total category. Functors lift to families in a pointwise fashion:

Problem 3.3.22. Given a 2-functor $F : \mathcal{C} \to \mathcal{D}$, lift it to a 2-functor $\mathsf{Fam}(F) : \mathsf{Fam}(\mathcal{C}) \to \mathsf{Fam}(\mathcal{D})$.

Construction. It suffices to define a 2-functor displayed over the identity 2-functor of the base hSet. Its action on objects, $\mathsf{Fam}_0(F) : \prod_{J:\mathsf{hSet}_0}(J \to \mathcal{C}_0) \to (J \to \mathcal{D}_0)$, is postcomposition with F_0. The action on morphisms and 2-cells are defined in the same way. They satisfy the laws of a displayed 2-functor by function extensionality. ⌟

Proposition 3.3.23. Lifting a 2-functor $F : \mathcal{C} \to \mathcal{D}$ inherits the following local properties:

1. If F is locally fully-faithful, then so is $\mathsf{Fam}(F)$.
2. If F is locally *split*-essentially surjective, then so is $\mathsf{Fam}(F)$.
3. Assume the axiom of choice for h-sets holds and that $\mathcal{C}$ is locally strict. If F is locally essentially surjective, then so is $\mathsf{Fam}(F)$.

Proof. Local fully-faithfulness follows from the pointwise definition of Fam_2: if $F_2 : \mathcal{C}_2(f, g) \to \mathcal{D}_2(F_1(f), F_1(g))$ is an equivalence, then so is postcomposition $\mathsf{Fam}_2(F) \doteq F_2 \circ \_$. Local split essential surjectivity follows from a similar pointwise argument.

In the non-split case, fix $X, Y : \mathsf{Fam}_0(\mathcal{C})$ and a morphism $(\psi, g) : \mathsf{Fam}_1(X, Y)$. It suffices to provide merely a family of sections,

$$\left\| \prod_{j:J} \sum_{f:\mathcal{C}_1(X_j, Y_{\psi j})} F_1(f) \cong g_j \right\|_{-1}$$

If F is locally essentially surjective, then in particular

$$\prod_{j:J} \left\| \sum_{f:\mathcal{C}_1(X_j, Y_{\psi j})} F_1(f) \cong g_j \right\|_{-1}$$

holds. We use choice for sets to move the truncation outward: J is a set, and so are $\mathcal{C}_1(X_j, Y_{\psi j})$ (by local strictness of $\mathcal{C}$) and local isomorphisms, $F_1(f) \cong g_j$. □

With this construction at hand, we can finally define the 2-category of action containers:

Definition 3.3.24. The 2-category of action containers is that of families of actions:

$$\mathsf{ActionCont} := \mathsf{Fam}(\mathsf{Action})$$ ⌟

By elementary manipulation of Σ and Π-types, we see that 0- and 1-cells in this 2-category coincide with the objects and morphisms of the 1-category of containers (cf. proposition 3.2.4). Explicitly, there are equivalences of types of objects

$$\begin{aligned}\mathsf{ActionCont}_0 &\simeq \sum_{S:\mathsf{hSet}_0} S \to \mathsf{Action}_0(S)\\ &\simeq \sum_{S:\mathsf{hSet}} S \to \sum_{G:\mathsf{Group}} \sum_{P:\mathsf{hSet}_0} (G \mathrel{\dot{\to}} \Sigma P)\\ &\simeq \sum_{S:\mathsf{hSet}} \sum_{P:S\to\mathsf{hSet}} \sum_{G:S\to\mathsf{Group}} \prod_{s:S} G_S \mathrel{\dot{\to}} \Sigma P_s\end{aligned}$$

and of morphisms from $F \doteq (S, \lambda s.(G_s, P_s, \sigma_s))$ to $F' \doteq (T, \lambda t.(H_t, Q_t, \tau_t))$,

$$\begin{aligned}\mathsf{ActionCont}_1(F,F') &\simeq \sum_{u:S\to T} \prod_{s:S} \sum_{\varphi:G_s \dot{\to} H_{us}} \sum_{f:P_s \leftarrow Q_{us}} \mathsf{isEquivariant}_{\sigma_s,\tau_{us}}(\varphi, f)\\ &\simeq \sum_{u:S\to T} \sum_{f:\prod_s Q_{us}\to P_s} \sum_{\varphi:\prod_s G_s \dot{\to} H_{us}} \prod_{s:S} \mathsf{isEquivariant}_{\sigma_s,\tau_{us}}(\varphi, f)\end{aligned}$$

If we apply similar transformations to the newly added type of 2-cells, we recover a familiar condition.

Proposition 3.3.25. Let $F, F' : \mathsf{Action}_0$, with their components denoted by $F \doteq (S, \lambda s.(G_s, P_s, \sigma_s))$ and $F' \doteq (T, \lambda t.(H_t, Q_t, \tau_t))$. For $u : S \to T$ and morphisms $f, g : \mathsf{ActionCont}_1(u; F, F')$, there is an equivalence

$$\mathsf{ActionCont}_2((u,f),(u,f')) \simeq \prod_{s:S} \sum_{r:\mathsf{Conjugator}(\varphi_s,\psi_s)} f'_s = g'_s \circ \tau_{us}(r),$$

in which $\varphi, \psi : \prod_s G_s \mathrel{\dot{\to}} H_{us}$ and $f', g' : \prod_s Q_{us} \to P_s$ are the maps of symmetries and positions of f and g, respectively.

Notice how the above resembles the quotient of premorphisms of quotient containers: the proposition $f'_s = g'_s \circ \tau_{us}(r)$ appears in the definition of $(u, f) \approx (u, f')$ in definition 3.1.6. In their original work, Abbott et al. add this condition

to ensure that the extension functor is well-defined, and argue that it is natural for container morphisms to be invariant under permutation of positions [Abb+04, Definition 4.1]. In our case it is necessary to characterize homotopies of bundle maps $\bar{\mathbf{B}}_1(f_s')$ and $\bar{\mathbf{B}}_1(g_s')$ (cf. theorem 3.3.20)

If we change our perspective from Bundle to SymmCont à la remark 3.3.15, then lifting the 2-functor $\bar{\mathbf{B}} : \mathsf{Action} \to \mathsf{Bundle}$ to families immediately gives a local characterization of action containers in terms of families of symmetric containers.

Corollary 3.3.26. The lifting $\mathsf{Fam}(\bar{\mathbf{B}}) : \mathsf{ActionCont} \to \mathsf{Fam}(\mathsf{SymmCont})$ is:

1. locally fully faithful,
2. assuming the axiom of choice, locally a weak equivalence.

Proof. Action is locally strict, and $\bar{\mathbf{B}}$ is locally fully-faithful and essentially surjective by theorem 3.3.20, so $\mathsf{Fam}(\bar{\mathbf{B}})$ inherits these properties by propositions 3.3.23.(1) and 3.3.23.(3), provided that choice holds for the latter. □

This says that constructively, morphisms of action containers correspond to a subcategory of morphisms of (families of) symmetric containers. Under the assumption of the axiom of choice, this construction covers all such morphism.

To connect action containers to symmetric ones, and not just families of the latter, note the following. Denote a family of bundles $(J, \lambda j.(B_j, F_j)) : \mathsf{Fam}(\mathsf{Bundle}_0)$ in *family notation* $\{B_j, F_j\}_{j:J}$. Any bundle can be considered as a single bundle by collecting all fibers over the sum of bases, $\sum_{j:J} B_j$. This "summation" extends to a 2-functor:

Problem 3.3.27. Define a 2-functor $\Sigma : \mathsf{Fam}(\mathsf{Bundle}) \to \mathsf{Bundle}$ with the following data:

- On objects, Σ_0 is given for any family of bundles $X \doteq \{B_j, F_j\}_{j:J}$ by

$$\Sigma_0(X) \coloneqq \left((\textstyle\sum_{j:J} B_j), \lambda(j,b).\, F_j(b) \right)$$

- The action Σ_1 maps families $f : \{B, F\}_J \to \{C, G\}_K$ to a pair of a reindexing function and a map of fibers. If $f \doteq (u, (\lambda j.(\varphi_j, e_j)))$, then

$$\Sigma_1(f) \coloneqq (\Sigma(u, \varphi), e^*),$$

where

$$\begin{array}{l} \Sigma(u,\varphi) : \sum_J B \to \sum_K C \\ \Sigma(u,\varphi) \coloneqq \lambda(j,b).\,(u(j), \varphi_j(b)) \end{array} \quad \text{and} \quad \begin{array}{l} e^* : \prod_{(j,b)} G_{uj}(\varphi_j(b)) \to F_j(b) \\ e^*((j,b), g) \coloneqq e_j(g) \end{array}$$

- On 2-cells, it takes a family of identifications of bundle maps to a identification of their sums via function extensionality.

Construction. Verifying that this is functorial is straightforward — the coherences hold almost definitionally. ⌟

As promised at the beginning of this section in diagram 3.2, turning action containers into symmetric ones now factors as follows:

Definition 3.3.28. Delooping of action containers is the composite 2-functor

$$\begin{array}{ccccc} \mathsf{ActionCont} & & \xrightarrow{\quad\mathbf{B}\quad} & & \mathsf{SymmCont} \\ {\scriptstyle =}\downarrow & & & & \uparrow{\scriptstyle =} \\ \mathsf{Fam}(\mathsf{Action}) & \xrightarrow[\mathsf{Fam}(\bar{\mathbf{B}})]{} & \mathsf{Fam}(\mathsf{Bundle}) & \xrightarrow[\Sigma]{} & \mathsf{Bundle} \end{array}$$

⌟

In particular, the action on objects and morphisms of this 2-functor coincide with our earlier "manual" definitions (definition 3.3.2 and problem 3.3.4) except that the data of containers is bundled slightly differently:

Corollary 3.3.29. On objects $\mathbf{B} : \mathsf{ActionCont} \to \mathsf{SymmCont}$ acts as

$$\mathbf{B}_0(\{G_s, P_s, \sigma_s\}_{s:S}) = \left((s,x) : \textstyle\sum_{s:S} \mathbf{B}G_s \lhd \bar{\mathbf{B}}\sigma_s(x)\right)$$

On a morphism $f : \mathsf{ActionCont}(F,G)$, we have $\mathbf{B}_1(f) = (\bar{\varphi}, \bar{f})$ where both $\bar{\varphi}$ and $\bar{f}$ are derived from the components of f like they are in problem 3.3.4.

We would like to show that the composite $\mathbf{B}$ is locally fully faithful. In this factorization, we understand $\mathsf{Fam}(\bar{\mathbf{B}})$, so now we turn to Σ. Unfortunately, Σ does not seem to be constructively locally fully faithful, at least not on all local categories. We can however establish this if we restrict ourselves to objects in the image of $\mathsf{Fam}(\bar{\mathbf{B}})$.

Lemma 3.3.30. Let $X, Y : \mathsf{Fam}_0(\mathsf{Bundle})$ with components $X \doteq \{B_j, F_j\}_{j:J}$. If the base B_j is a connected type for all $j : J$, then the local functor $\Sigma_1 : \mathsf{Fam}_1(X,Y) \to \mathsf{Bundle}(\Sigma_0(X), \Sigma_0(Y))$ is fully faithful.

Proof. We have to show that for morphisms $(u,f),(v,g) : \mathsf{Bundle}_1(X,Y)$, the action on 2-cells

$$\Sigma_2 : (u,f) \Rightarrow (v,g) \to \Sigma_1(u,f) \Rightarrow \Sigma_1(v,g)$$

is an equivalence. We do so by decomposing Σ_2 into a sequence of maps, of which all but one are equivalences by basic properties of Σ, Π, and $=$-types. It remains to show that, for all $j : J$, the constant map

$$\begin{aligned} \mathsf{c}_A &: A \to (B_j \to A) \\ \mathsf{c}_A &\coloneqq \lambda a.\, \lambda b.\, a \end{aligned}$$

is an equivalence for $A \coloneqq u(j) = v(j)$. But c_A fully characterizes connectedness: B_j is connected if and only if c_A is an equivalence for all h-sets A [Uni13, Corollary 7.5.9]. In our case $u, v : J \to K$ are maps between the sets indexing X and Y, respectively. Therefore, $u(j) = v(j)$ is a proposition (hence a set), so $\mathsf{c}_{u(j)=v(j)}$ is necessarily an equivalence. For details of the proof, refer to [IIa, `GpdCont.SetBundle.Summation`]. □

This classifies the local behaviour of the inclusion of action containers in symmetric containers.

Theorem 3.3.31. The composite 2-functor

$$\mathbf{B} \coloneqq \Sigma \circ \mathsf{Fam}(\bar{\mathbf{B}}) : \mathsf{ActionCont} \to \mathsf{SymmCont}$$

is locally fully faithful.

Proof. It suffices to show that local functors of both $\mathsf{Fam}(\bar{\mathbf{B}})$ and Σ are fully-faithful. The former is so by corollary 3.3.26.(1). For the latter, we only need to verify this on the image of $\mathsf{Fam}_1(\bar{\mathbf{B}})$, where we can employ lemma 3.3.30: By construction, the base of an associated bundle is connected – after all, it is a delooping of some group. □

With this, we have achieved the goal set out at the beginning of this section: Not only is the obvious translation from one type of containers to the other functorial, it also embeds morphisms as a category, preserving interesting 2-dimensional structure.

3.4 A 1-category of symmetric containers

Although the construction of a subcategory of symmetric containers is interesting, the data contained in the 2-cells of action containers could be regarded as an artifact of the encoding. If containers are to be interpreted as datatypes, with morphisms representing polymorphic functions between them, then a conjugator expresses a notion of proof-relevant program equivalence: two functions are equivalent if they only differ by some permutation of the data they transform. While this is undoubtedly a useful notion when proving properties of such

programs, we are curious to see if the heavy machinery of 2-categories can be avoided; at least if one is only interested in program construction.

In this section, we show that symmetric containers form a 1-category, as long as groupoids of shapes are equipped with the structure of a *skeleton*. This structure decomposes each shape-groupoid into a family of groups; we prove that the category of symmetric containers with skeletal shapes is equivalent to the 1-category of action containers as given by proposition 3.2.4.

That this might be possible is guided by the following observation: The image on objects of the 2-functor $\mathbf{B} : \mathsf{Group} \to \mathsf{hGrpd}$ is not only groupoids, but *pointed*, *connected* groupoids. We have already exploited this, for example, in lemma 3.3.30. This defines a 2-category of *structured h*-groupoids:

Definition 3.4.1. The 2-category hGroup displayed over hGrpd, consists of the following data:

- Over a groupoid G, displayed objects are a choice of *point* in G, and a proof that G is *connected*:
$$\mathsf{hGroup}_0(G) :\equiv \sum_{\mathsf{pt}_G : G} \mathsf{isConnected}(G)$$
- Morphisms over some $f : G \to H$ are sets of identifications of points,
$$\mathsf{hGroup}_1\left(f ; (\mathsf{pt}_G, \_), (\mathsf{pt}_H, \_)\right) :\equiv \left(f(\mathsf{pt}_G) = \mathsf{pt}_H\right)$$
- Over a 2-cell in hGrpd, i.e. a homotopy $p : f = f'$, 2-cells between $q : f(\mathsf{pt}_G) = \mathsf{pt}_H$ and $f'(\mathsf{pt}_G) = \mathsf{pt}_H$ are dependent paths
$$\mathsf{hGroup}_2(p; q, q') :\equiv (q =^Y_p q')$$
where $Y(f : G \to H) :\equiv f(\mathsf{pt}_G) = \mathsf{pt}_H$. ⌟

In this 2-category, each groupoid carries the structure of a point, which is preserved by morphisms. The types of displayed 2-cells are propositions — they are paths between homotopies of maps, and those homotopies form sets. It is natural to ask: what *are* the homotopies of pointed maps between pointed, connected groupoids? That is, what are the 2-cells of the *total* category hGroup, exactly? Surprisingly, the extra requirement of having to preserve basepoints forces 2-cells to be unique, if they exist:

Proposition 3.4.2 ([Bez+24, Lemma 4.4.12])**.** The 2-category hGroup is locally thin. This implies, in particular, the following:

1. Morphisms $\mathsf{hGroup}_1((G, \mathsf{pt}_G), (H, \mathsf{pt}_H))$ form a set.

2. hGroup forms a 1-category. □

By a slight modification of the proofs in § 3.3.1, we can show that hGroup is, in fact, *equivalent* to the 1-category of ordinary groups.

Proposition 3.4.3. There is an equivalence of categories $\mathsf{Group} \simeq \mathsf{hGroup}$ whose underlying functors are induced by delooping $\mathbf{B} : \mathsf{Group} \to \mathsf{hGroup}$ and the loop space operator $\Omega : \mathsf{hGroup} \to \mathsf{Group}$, respectively.

Proof. We can turn the local weak equivalence of theorem 3.3.12 into an isomorphism of hom-sets by using the extra structure provided by the 1-cells in hGroup. Part of the data of a 1-cell $(f, p) : \mathsf{hGroup}_1((G, \mathsf{pt}_G), (H, \mathsf{pt}_H))$ is a chosen path $p : f(\mathsf{pt}_G) = \mathsf{pt}_H$. Inspecting the proof of proposition 3.3.11, we see that this lets us construct an explicit inverse $\mathsf{hGroup}_1(\mathbf{B}G, \mathbf{B}H) \to \mathsf{Group}_1(G, H)$. From this it follows that $\mathbf{B}$ is fully faithful. A straightforward argument shows that it is *split* essentially surjective as well, from which we conclude that it is an equivalence of categories. □

This more than justifies the name hGroup: just as categorically, single-object groupoids are groups, from a univalent perspective, pointed, connected h-groupoids are "h-groups". This perspective on groups is explored to great effect in *Symmetry* [Bez+24] by Bezem et al., where objects in Group are *abstract groups*, and those in hGroup are *concrete*.

3.4.1 The 1-category of skeletal groupoids

Delooping an action container results in a groupoid of shapes that is a sum of h-groups, that is, a set-indexed sum of pointed, connected groupoids. In the other direction, however, it does not follow that an arbitrary groupoid of shapes arises from a family of groups. Even though in general, any $A : \mathsf{hGrpd}$ is a set-indexed sum of connected h-groupoids (cf. proposition 1.4.15) it is not clear why these h-groupoids should be *pointed*. A skeleton is exactly the missing structure that turns an h-groupoid into a sum of h-groups:

Definition 3.4.4. A *skeleton* on $A : \mathsf{Type}$ is a choice of basepoint for all connected components of A, i.e. a section to the truncation map:

$$\mathsf{Skel}(A) :\equiv \sum_{\mathsf{pt}:\|A\|_0 \to A} |\_|_0 \circ \mathsf{pt} = \mathsf{id}_{\|A\|_0}$$

We say that A is *skeletal* if it carries the structure of a skeleton. ⌟

The following are sometimes useful perspectives on skeletons:

Lemma 3.4.5. The structure of a skeleton on $A : \mathsf{Type}$ is equivalently one of the following:

1. Inhabited $|\_|_0$-fibers:

$$\mathsf{Skel}(A) \simeq \prod_{x:\|A\|_0} \mathsf{fiber}_{|\_|_0}(x)$$

2. Mere retractions of $|\_|_0$:

$$\mathsf{Skel}(A) \simeq \sum_{\mathsf{pt}:\|A\|_0 \to A} \prod_{a:A} \|\mathsf{pt}\,|a|_0 = a\|_{-1}$$

Proof. By rearranging quantifiers, and the fact that $(|a|_0 = |b|_0) \simeq \|a = b\|_{-1}$ for all $a, b : A$. □

The first says that a skeleton is exactly a proof that the surjection $|\_|_0$ *splits*, resembling a choice principle. The second says that any point is merely equal (or, *connected*) to the chosen basepoint in its component. In order to establish properties of skeletal types, it suffices to do so on the basepoints:

Proposition 3.4.6. For any $A : \mathsf{Type}$ with skeleton $\mathsf{pt}_A : \|A\|_0 \to A$ and family $P : A \to \mathsf{Prop}$, there is an equivalence

$$\Big(\prod_{x:\|A\|_0} P(\mathsf{pt}_A(x))\Big) \simeq \Big(\prod_{a:A} P(a)\Big)$$

In the following, we write $\|\varphi\|_0 : \|G\|_0 \to \|H\|_0$ instead of $\mathsf{map}_{|\_|_0}(\varphi)$ for truncation of a map $\varphi : G \to H$. The correct notion of morphism of skeletal types is a map that preserves this structure:

Definition 3.4.7. Given skeletal types (G, pt_G) and (H, pt_H), a map $\varphi : G \to H$ is *skeleton-preserving* if there is a filler making the square

$$\begin{array}{ccc} \|G\|_0 & \xrightarrow{\mathsf{pt}_G} & G \\ {\scriptstyle \|\varphi\|_0}\downarrow & & \downarrow{\scriptstyle \varphi} \\ \|H\|_0 & \xrightarrow[\mathsf{pt}_H]{} & H \end{array}$$

commute, i.e. one can give a path $\varphi_\square : \varphi \circ \mathsf{pt}_G = \mathsf{pt}_H \circ \|\varphi\|_0$. Note that the filler $\varphi_\square$ need not be unique; in general homotopies of maps $\|G\|_0 \to H$ form an h-set. ⌟

Skeletons and maps preserving them define a wild category displayed over Type; when restricted to h-groupoids, this induces a 2-category of skeletal groupoids:

Proposition 3.4.8. Skeletal groupoids and skeleton-preserving maps define a 2-category SkGpd.

Proof. By lemma 1.5.17: between h-groupoids, skeleton-preserving maps form an h-groupoid. □

Just like in the case of hGroup, assuming that morphisms preserve skeletons is enough to trivialize the 2-dimensional structure of this category:

Theorem 3.4.9. For all $G, H : \mathsf{SkGpd}_0$, the type $\mathsf{SkGpd}_1(G,H)$ is an h-set.

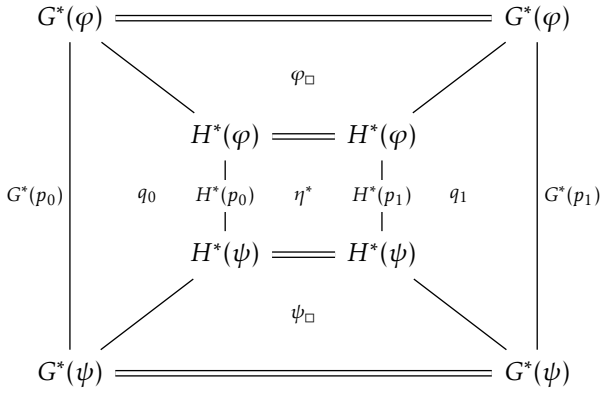
Proof. Let G, H be skeletal groupoids. For all $(\varphi, \varphi_\square), (\psi, \psi_\square) : \mathsf{SkGpd}_1(G,H)$ define

$$\mathsf{Code} := \sum_{p:\varphi=\psi} \varphi_\square =_p \psi_\square$$

Evidently, $((\varphi, \varphi_\square) = (\psi, \psi_\square)) \simeq \mathsf{Code}$, hence it suffices to show that Code is a proposition. Over any $p : \varphi = \psi$, the type $\varphi_\square =_p \psi_\square$ is a proposition — both $\varphi_\square$ and $\psi_\square$ are homotopies of maps into H, which is an h-groupoid. Therefore, given $(p_0, q_0), (p_1, q_1) : \mathsf{Code}$, it is enough to give some $\gamma : p_0 = p_1$. By proposition 3.4.6 and function extensionality, it suffices to do this pointwise; there is an equivalence

$$(p_0 = p_1) \simeq (\mathsf{cong}_{G^*}(p_0) = \mathsf{cong}_{G^*}(p_1)) \quad \text{where } G^*(\rho : G \to H) := \rho \circ \mathsf{pt}_G$$

We define $\gamma^* : \mathsf{cong}_{G^*}(p_0) = \mathsf{cong}\, G^*(p_1)$ (a path between paths) as a composite square



in which $H^*(\rho : G \to H) :\equiv \mathsf{pt}_H \circ \|\rho\|_0$, and double-dashed arrows are refl-paths. The terms $\varphi_\square$ and $\psi_\square$ fill the top- and bottom squares

$$\begin{array}{ccc} \varphi \circ \mathsf{pt}_G & = & \varphi \circ \mathsf{pt}_G \\ \| & \varphi_\square & \| \\ \mathsf{pt}_H \circ \|\varphi\|_0 & = & \mathsf{pt}_H \circ \|\varphi\|_0 \end{array} \quad \text{and} \quad \begin{array}{ccc} \mathsf{pt}_H \circ \|\psi\|_0 & = & \mathsf{pt}_H \circ \|\varphi\|_0 \\ \| & \psi_\square & \| \\ \psi \circ \mathsf{pt}_G & = & \varphi \circ \mathsf{pt}_G \end{array}$$

whereas q_0 and q_1 identify $\varphi_\square$ and $\psi_\square$ over p_0 and p_1, respectively. To exhibit the missing η^*, notice that H^* factors through an h-set, that is

$$\begin{array}{ccccc} G \to H & & \xrightarrow{\quad H^* \quad} & & \|G\|_0 \to H \\ & \searrow^{\|\_\|_0} & & \nearrow_{\mathsf{pt}_H \circ \_} & \\ & & \|G\|_0 \to \|H\|_0 & & \end{array}$$

As such it is weakly constant on paths — in particular on p_0 and p_1 — hence we get some $\eta^* : \mathsf{cong}_{H^*}(p_0) = \mathsf{cong}_{H^*}(p_1)$. □

Theorem 3.4.10. The objects and morphisms of SkGpd form a 1-category, denoted by the same name.

Proof. Composition of skeleton-preserving maps is unital and associative, up to a path. But by theorem 3.4.9 paths between morphisms are propositions, hence objects and morphisms form a 1-category. □

The proof of theorem 3.4.9 is essentially a many-component version of proposition 3.4.2, and h-groups naturally embed into skeletal groupoids:

Proposition 3.4.11. The types of h-groups and connected skeletal groupoids are equivalent. In particular, hGroup is a full subcategory of SkGpd.

Proof. Let G be a connected groupoid. It suffices to show that the type of skeletons $\mathsf{Skel}(G)$ is equivalent to the type of h-group structures on G, i.e. a choice of point in G. By lemma 3.4.5.(1),

$$\mathsf{Skel}(G) \simeq \prod_{x : \|G\|_0} \mathsf{fiber}_{|\_|_0}(x)$$

But G is connected, hence $\|G\|_0$ is contractible with center $c_0 : \|G\|_0$. Therefore,

$$\mathsf{Skel}(G) \simeq \mathsf{fiber}_{|\_|_0}(c_0) \simeq \sum_{g:G} |g|_0 =_{\|G\|_0} c_0 \simeq G$$

By the same argument we see that skeleton-preserving maps between connected groupoids are exactly pointed maps, i.e. morphisms of h-groups: If G and H are both connected then $\varphi : G \to H$ preserves their skeletons if and only if

$$\begin{array}{ccc} 1 & \xrightarrow{\mathsf{pt}_G} & G \\ \downarrow & & \downarrow{\scriptstyle \varphi} \\ 1 & \xrightarrow[\mathsf{pt}_H]{} & H \end{array}$$

commutes, that is, whenever φ is a pointed map. This exhibits hGroup as a full subcategory of all skeletal groupoids. □

Skeletal groupoids are closed under Σ, as long as the sum is over an h-set. Consequently, SkGpd has all coproducts.

Lemma 3.4.12. For all $J : \mathsf{hSet}$ and $G : J \to \mathsf{SkGpd}$, $\sum_{j:J} G_j$ has a skeleton.

Proof. Since J is a set, we can define a map

$$\left\| \sum_{j:J} G_j \right\|_0 \xrightarrow{\simeq} \sum_{j:J} \|G_j\|_0 \xrightarrow{\Sigma(\mathsf{id},\mathsf{pt}_G)} \sum_{j:J} G_j$$

It is straightforward to show that this is a section of the truncation map by showing that $\Sigma(\mathsf{id}, \mathsf{pt}_G)$ is a section of $\Sigma(\mathsf{id}, |\_|_0) : \sum_{j:J} G_j \to \sum_{j:J} \|G_j\|_0$. □

Proposition 3.4.13. SkGpd admits all set-indexed coproducts.

Proof. Given a family $G : J \to \mathsf{SkGpd}_0$ over some $J : \mathsf{hSet}$, the sum $\sum_{j:J} G_j$ admits a skeleton by lemma 3.4.12. The inclusion maps $\iota_j : G_j \to \sum_J G$ are skeleton-preserving. A routine argument shows that this defines coproducts on SkGpd. □

This implies that skeletal groupoids are nothing but families of groups; not just on the level of objects, but as categories:

Theorem 3.4.14. SkGpd is the free coproduct completion of hGroup: the subcategory inclusion $\mathsf{hGroup} \hookrightarrow \mathsf{SkGpd}$ of proposition 3.4.11 extends to an equivalence of 1-categories

$$\Sigma : \mathsf{Fam}(\mathsf{hGroup}) \simeq \mathsf{SkGpd}$$

Proof. By proposition 3.4.13, SkGpd has all coproducts. By the universal property of Fam as a coproduct-completion, this defines a functor $\Sigma : \mathsf{Fam}(\mathsf{hGroup}) \to \mathsf{SkGpd}$. In the other direction, any skeletal groupoid is the sum of its (pointed, connected) components, hence forms a family of groups. In the same way, any skeleton-preserving map is just a family of pointed maps, indexed by components. This extends to a functor, and showing that both functors together form an equivalence is straightforward, albeit tedious. □

3.4.2 Bundles over skeletal groupoids

In order to show that skeletal symmetric containers and action containers coincide, we have to ensure that, just like in § 3.3, families of actions correspond to bundles. Bundles over skeletal groupoids are bundles over the underlying groupoids, so the 2-category Bundle of problem 3.3.14 immediately restricts to a 1-category:

Proposition 3.4.15. The total 2-category

$$\mathsf{SkBundle} \coloneqq \int_{(G,\mathrm{pt}_G):\mathsf{SkGpd}} \mathsf{Bundle}(G)$$

restricts to a 1-category of bundles over skeletal groupoids.

Proof. By definition, morphisms of bundles are just maps into h-sets. Together with theorem 3.4.9, morphisms of its total category form a set. Composition is unital and associative up to a path, which suffices to conclude that SkBundle is a 1-category. □

Summation of bundles like in problem 3.3.27 shows that SkBundle has 1-categorical coproducts. As such, we can follow the proof-technique of theorem 3.4.14 to show that bundles over skeletal groupoids are exactly families of group actions:

Theorem 3.4.16. The 2-functor taking associated bundles $\bar{\mathbf{B}} : \mathsf{Action} \to \mathsf{Bundle}$ (definition 3.3.19) restricts to a fully-faithful functor $\mathsf{Action} \to \mathsf{SkBundle}$. This induces an equivalence of 1-categories $\mathsf{Fam}(\mathsf{Action}) \simeq \mathsf{SkBundle}$.

Proof. By theorem 3.3.20, $\bar{\mathbf{B}}$ is locally an equivalence of 2-categories. The same proof shows that it is fully-faithful if restricted to the 1-categories Action and SkBundle. The codomain SkBundle has coproducts, hence there is some $F : \mathsf{Fam}(\mathsf{Action}) \to \mathsf{SkBundle}$. Both SkBundle and Fam(Action) are univalent categories, so by proposition 1.5.10, this functor is an equivalence of categories if it is fully-faithful and an equivalence on objects.

On objects, F acts like the 2-functor $\Sigma \circ \mathsf{Fam}(\bar{\mathbf{B}}) : \mathsf{Fam}(\mathsf{Action}) \to \mathsf{Bundle}$; it sends a family $\{G_s, P_s, \sigma_s\}_{s:S}$ to the bundle whose (skeletal) base is $\sum_s \mathbf{B}G_s$, and fibers are $\bar{\mathbf{B}}\sigma_s$ (see corollary 3.3.29). For any $(B, F) : \mathsf{SkBundle}$, there exists a unique family of groups $\{G_s\}_{s:S}$ such that $B \simeq \sum_s \mathbf{B}G_s$ by the equivalences of proposition 3.4.3 and theorem 3.4.14. By univalence, this extends to an equivalence of bundles, hence $F_0 : \mathsf{Fam}_0(\mathsf{Action}) \to \mathsf{SkBundle}_0$ is an equivalence.

By theorem 3.3.31, F is faithful; that it is also full follows because its action on maps creates skeleton-preserving maps. □

We alias the 1-category SkSymmCont := SkBundle, and say *skeletal container*[5] to mean symmetric container with skeletal shapes. Changing our point of view from bundles to containers we obtain the following characterization of action containers:

Theorem 3.4.17. Action containers and symmetric containers with skeletal shapes are, by univalence, propositionally equal categories. □

By transporting the extension functor from ActionCont, we get a functor $[\![\_]\!] : \mathsf{SkSymmCont} \to \mathsf{Endo}(\mathsf{hSet})$. Notably, computing its action on objects amounts to an application of problem 3.1.24 (which is valid even for containers whose actions are not faithful), and reduces to

$$[\![(S, \mathrm{pt}_S) \triangleleft P]\!](X) \simeq \Big\| \sum_{s:S} (P_s \to X) \Big\|_0 \tag{3.3}$$

The extension of a skeletal container is simply the truncated extension of the underlying symmetric container.

This equivalence of the *abstract* (action containers) and the *concrete* (symmetric containers with a skeleton) lets us benefit from either. In practice symmetric containers are skeletal, not just at an intuitive level, but because we can always define them in terms of action containers. In the other direction, we can still use the machinery of higher inductive types to define groupoids of shapes and, coherently, families of positions. As long as we can find a skeleton of these shapes, we can present the symmetries of this container as hands-on groups. Furthermore, extension (equation (3.3)) gives a 1-categorical interpretation as hSet-endofunctors, at the cost of identifying some container morphisms as the same natural transformation.

3.4.3 Towards substitution

So far, we have avoided deriving a *substitution* for action containers. Curiously, it is not defined for quotient containers either, despite it being easy to define for ordinary containers. Let us define substitution at least for some classes of action containers. We view them as skeletal containers, and identify sufficient conditions such that for skeletal $F, G : \mathsf{SymmCont}$, the substitution $F[G]$ is again skeletal.

Recall from definition 1.6.5, that for ordinary containers $F \doteq (S \triangleleft P)$ and $G \doteq (T \triangleleft Q)$, substitution $F[G] : \mathsf{Cont}$ is defined as

$$(F[G])_{\mathsf{Sh}} :\equiv \sum_{s:S} (P_s \to T) \qquad \text{and} \qquad (F[G])_{\mathsf{Ps}}(s, f) :\equiv \sum_{p:P_s} Q_{f(p)}$$

[5] We refrain from calling them *coffins*.

This definition respects the truncation level of either container; if both F and G are (n,k)-truncated, then so is $F[G]$ — in its shapes, P_s appears in a negative position, hence does not influence their truncation level. Furthermore, $\_[\_]$ is a monoidal operation of containers with unit $\mathsf{Id} \doteq (1 \lhd 1)$. By univalence, there are identifications of containers $F[\mathsf{Id}] = F$ and $\mathsf{Id}[G] = G$, as well as $(F[G])[H] = F[(G[H])]$. This defines a (non-symmetric) monoidal structure on the 2-category of symmetric containers which corresponds to composition of extensions, i.e. $[\![F[G]]\!] = [\![F]\!] \circ [\![G]\!]$. If action containers are closed under substitution, then so are skeletal containers, and this structure should be preserved by the forgetful 2-functor $\mathsf{SkSymmCont} \to \mathsf{SymmCont}$. Therefore, whether action containers admit a substitution operation reduces to the following question: If F and G are skeletal containers, is $F[G]$ as well? In particular, if we are given skeletons on the shape groupoids S and T, does $\sum_{s:S}(P_s \to T)$ admit a skeleton?

In the previous section we have seen that skeletal types are closed under sums as long as the sum is indexed by an h-set (lemma 3.4.12). Here, however, we are asking whether a sum indexed by the h-groupoid S has a skeleton. In specific instances, this is the case; in particular if the groupoids involved are groups:

Problem 3.4.18. For h-groups S, T and a family $P : S \to \mathsf{hSet}$ of *finite* sets, the type $\sum_{s:S}(P_s \to T)$ is an h-group, hence has a skeleton.

Construction. The sum is clearly a groupoid, and pointed by $(\mathsf{pt}_S, \lambda\_.\, \mathsf{pt}_T)$. It remains to show that it is a connected type. But finite products of connected types are connected, hence $P_s \to T$ is connected. Similarly, sums of connected types are connected, thus $\sum_{s:S}(P_s \to T)$ is connected. ⌟

The above combines two group-theoretic constructions in a type-theoretic formulation: The direct product $\prod_{j:J} G_j$ of finitely many groups, and the *semidirect product*:

Definition 3.4.19 ([Bez+24, § 7.2])**.** For $G : \mathsf{hGroup}$ and a family $H : G \to \mathsf{hGroup}$, the (connected) groupoid $\sum_{g:G} H(g)$ is pointed by $(\mathsf{pt}_G, \mathsf{pt}_{H(\mathsf{pt}_G)})$. The resulting group, $G \ltimes H : \mathsf{hGroup}$ is the *semidirect product* of G and H. ⌟

A special case of semidirect product is that of a *wreath product*:

Definition 3.4.20 ([Bez+24, § 7.3])**.** Let $G, H : \mathsf{hGroup}$ and $X : G \to \mathsf{FinSet}$ an action of G on a finite set. Then

$$H \wr_X G := (g : G) \ltimes (X(g) \to H)$$

is called the *wreath product* of H by G via X. ⌟

Thus problem 3.4.18 can be rephrased as follows:

Proposition 3.4.21. Let $(S \lhd P), (T \lhd Q) : \mathsf{SkSymmCont}$. If S and T are connected and P is a family of finite sets, then $(S \lhd P)[(T \lhd Q)] : \mathsf{SymmCont}$ is a skeletal container. The skeleton on shapes is the wreath product $T \wr_P S : \mathsf{hGroup}$. □

Skeletal symmetric containers with connected shapes correspond exactly to group actions by theorem 3.4.16, which in turn are nothing but single-shape action containers. Consequently, single-shape action containers with finite positions are closed under a substitution operation.

We can generalize the semidirect product slightly, and define a skeleton on a sum indexed by a skeletal groupoid:

Proposition 3.4.22. For $A : \mathsf{SkGpd}$ and $H : A \to \mathsf{hGroup}$, the groupoid $\sum_{a:A} H(a)$ has a skeleton.

Proof. For all $a : A$, the type $\|H(a)\|_0$ is contractible, hence we obtain a family of points

$$\mathsf{pt}_{\Sigma_A B} : \Big\|\sum_{a:A} H(a)\Big\|_0 \xrightarrow{\simeq} \Big\|\sum_{a:A} \|H(a)\|_0\Big\|_0 \xrightarrow{\simeq} \|A\|_0 \xrightarrow{\mathsf{pt}} \sum_{a:A} H(a)$$

where $\mathsf{pt}(x) :\equiv (\mathsf{pt}_A(x), \mathsf{pt}_{H(\mathsf{pt}_A(x))})$. It follows immediately that $\mathsf{pt}_{\Sigma_A B}$ is a section of the truncation map. □

As an immediate consequence, we see that substitution of single-shape action containers into *any* action container with finite positions is well-defined:

Corollary 3.4.23. Let $(S \lhd P), (T \lhd Q) : \mathsf{SkSymmCont}$. If T is connected and P is a family of finite sets, then $(S \lhd P)[(T \lhd Q)] : \mathsf{SymmCont}$ is a skeletal container.

Proof. By proposition 3.4.22, the shapes of the substitution are skeletal. □

It is possible to substitute into a non-finitary container as long as the positions of the outer container satisfy the axiom of choice.

Lemma 3.4.24. If $J : \mathsf{hSet}$ satisfies $\mathsf{AC}_{0,-1}$, then $\prod_{j:J} G_j$ is an h-group for any family $G : J \to \mathsf{hGroup}$.

Proof. It suffices to show that the product $\prod_{j:J} G_j$ is connected. This is the case if $\|f = f'\|_{-1}$ for all $f, f' : \prod_{j:J} G_j$. Since G_j is connected, we know that $\forall j. \|f(j) = f'(j)\|_{-1}$. By an application of choice, we obtain $\|\forall j. f(j) = f'(j)\|_{-1}$, hence f are f' are merely equal by function extensionality. □

Corollary 3.4.25. Let $(S \lhd P), (T \lhd Q) : \mathsf{SkSymmCont}$. If T is connected and $P(s)$ satisfies the axiom choice $\mathsf{AC}_{0,-1}$ for all $s : S$, then $(S \lhd P)[(T \lhd Q)] : \mathsf{SymmCont}$ is a skeletal container.

Proof. By lemma 3.4.24, $P(s) \to T$ is an h-group, so proposition 3.4.22 applies. □

The appearance of a wreath product in proposition 3.4.21 is not entirely surprising. In combinatorics, wreath products describe symmetries of nested structures. Consider for example the square □ on which a group R acts by rotation. The group that acts by individually rotating n copies of □ is $R \wr_{[n]} 1$, which is equivalent to the direct product R^n. If in addition to rotating one wants to permute the boxes among themselves, then this group becomes $R \wr_{[n]} \Sigma_n$. Notice that Σ_n is the group of symmetries of an n-element *bag*, so the differences between $R \wr_{[n]} 1$ and $R \wr_{[n]} \Sigma_n$ is whether we substituted into an ordered or unordered collection. Similarly, wreath products conveniently describe symmetries of trees — see [Wel76] for a general overview of wreath products, and § 15 therein for their application to trees. A similar pattern has been noticed by Hasegawa when describing the composition of analytic functors [Has02]. In chapter 5 we will discuss how their presentation might help find a general substitution for action containers.

3.5 Conclusion

In this chapter we have seen that any quotient container gives rise to an ordinary container if we allow its shapes to be an h-groupoid. Both containers have the same extensions, as long as we interpret these as set-endofunctors. We notice that the same lifting also applies to action containers whose symmetries of positions do not act faithfully. By promoting a quotient of morphisms to a proof-relevant relation, we embed the resulting 2-category of action containers into that of symmetric containers, ensuring that this translation does not lose any information.

Either side of this translation has its purpose. Action containers are a useful language for specifying basic types with a wide variety of symmetries. They are entirely set-based and shapes, positions, and symmetries are specified separately. This makes it easier to construct specific examples if one is not comfortable defining shapes and symmetries together in a single h-groupoid. In practice, this is how symmetric containers are given anyways,[6] and theorem 3.4.17 ensures that all symmetric containers arise like that, assuming that we can identify the connected components of their shapes. However, it seems like action containers (and therefore quotient containers) do not not readily admit a substitution operation. This makes it impossible to compute presentations of composite datatypes, which is otherwise a basic requirement for any system aiming to capture the idea of polymorphic datatypes. Symmetric containers, on the other hand,

[6] In particular, all examples of [Gyl11, § 3.1] are of this kind.

compose trivially, and are in general much easier to reason about. Internalizing symmetric containers from "set-valued functors over groupoids" to "set-bundles over h-groupoids", like we did, aligns them nicely with the (homotopy) type-theoric primitives. Any construction on symmetric containers automatically preserves symmetries simply because it has to be homotopy-coherent, that is, has to preserve path-types.

There is hope to recover explicit presentations of symmetries for composite types by the results of § 3.4, such as corollaries 3.4.23 and 3.4.25. These results are not exhaustive, but point to a strategy: if one can find assumptions on skeletal containers F and G strong enough to construct a skeleton on the shapes of $F[G]$, then one can represent symmetries for any shape as some group.

4 Containers in Higher Types

In the previous chapters we have argued that, although set-based definitions of containers can present datatypes with symmetries, it is better to embrace the higher structure of types afforded by our univalent foundations, and to study containers in groupoids. Much of the structure of ordinary, set-truncated containers is well-behaved for symmetric containers. This applies in particular to substitution _[_], which is cumbersome, if not impossible, to describe for set-based approximations of datatypes with symmetries. In this chapter, we go a step further and drop the truncation assumption entirely, and study arbitrary Type-valued containers instead.

The main contribution of this chapter lies in § 4.5: we define a *derivative operation* ∂ in the sense of [Abb+05], but for arbitrary Type-valued containers. Traditionally, derivatives of containers require positions to have decidable equality [Abb+05, Theorem 5.1]. By *Hedberg's Theorem* [Hed98], it seems like such a derivative would force Type-valued containers to have set-truncated positions. We circumvent this apparent restriction by defining our derivative with respect to *isolated* positions.

We recall the concept of *isolated points* of a type in § 4.4 following ideas by Kraus et al. [Kra+17]. We give a number of, to our knowledge, novel results, including a characterization of their closure under dependent sums (theorem 4.4.15) and an elimination principle for types with a chosen isolated point (proposition 4.4.26). These results are necessary to ensure that the derivative is well-behaved, but otherwise independent of our study of containers. We hope to present them in a way that is useful in other contexts.

In § 4.5.1, we use the properties of isolated points to ensure that ∂ interacts with the structure of containers as expected: constant containers have constant (zero) derivative, the derivative of a sum is the sum of its derivatives, and derivatives of products follow a Leibniz rule. In other words, derivative behave like their analytic namesakes, even for containers of arbitrary types.

In § 4.5.2, we ensure that ∂ admits its expected universal property; namely that *cartesian* container morphisms from F into ∂G are exactly cartesian mor-

phisms out of $F \otimes \mathsf{Id}$. This captures the idea that ∂G is " G with a hole " by means of a mapping-in property. In order to state this property, we start this chapter by introducing the *wild category* of Type-valued containers in § 4.1. In problem 4.5.11, we prove that ∂ is right-adjoint to $\_ \otimes \mathsf{Id}$ in the wild wide subcategory of containers and cartesian morphisms. As a direct consequence we extend the universal property of the traditional derivative à la [Abb+05] from *discrete* to *all set-truncated* containers (theorem 4.5.12).

In § 4.5.4, we establish a *lax chain rule*: we construct a canonical morphism into the derivative of a substitution and show that this morphism is, constructively, an embedding of containers. For discrete containers, this chain rule is *strong*, i.e. an isomorphism of containers. For general containers, however, this is not the case: an invertible chain rule is *inconsistent* if assumed for arbitrary containers (theorem 4.5.25), or at least constructively taboo (corollary 4.5.27) when restricted to set-truncated containers. We derive a directed rule for derivatives of least container fixpoints from the chain rule, which is subject to the same caveats (theorem 4.5.40).

In order to ensure that our notion of container-fixpoint is correct, we take a slight detour in § 4.3 and prove that Altenkirch et al.'s construction of a fixpoint container μF [Alt+15] is valid even in the absence of *uniqueness of identity proofs*: We give an initial object in the wild category of $F[\_]$-algebras for *any* indexed, Type-valued container F. This construction is closely related to that of initial- and terminal container *functors* of Damato et al. [DAL25]; however we avoid possible coherence-issues of $[\![\_]\!]$ by constructing the fixpoint directly in the wild category of containers. We believe that the same strategy can be used to construct largest fixpoints νF from M-types, which are derivable from W-types in HoTT thanks to [ACS15].

From now on we refer to Type-valued containers simply as *containers*. If necessary, we qualify further and explicitly talk about e.g. *truncted* containers.

4.1 The wild category of Type-valued containers

Reasoning about containers in Type feels just like it does in h-sets. What changes, however, is that commutative diagrams of morphisms of Type-valued containers are now *structure*: Even though composition of morphisms is still unital and associative up some chosen identifications, these identifications need not be unique. Of course, the default unitors and associators are still canonical in some sense, but in order to express that, we need to talk about identifications of identifications of morphisms. This tower of coherences keeps growing, and as such, Type-valued container do not form 1-category, 2-category, etc. Intuitively, this "∞-groupoid" structure of morphisms presents containers as an $(\infty,1)$-category. Unfortunately, expressing internally to homotopy type theory exactly

what we mean by "$(\infty,1)$-category" is a famously difficult problem.[1] Instead, we are going to introduce the *wild category* of Type-valued containers as an approximation.

Problem 4.1.1. Define Cont, the wild category of containers and their morphisms. We write $F \to G$ for the type of container morphisms, $\mathsf{id}_F : F \to F$ for the identity morphisms, and $f ; g : F \to H$ for the diagrammatic composite of $f : F \to G$ and $g : G \to H$.

Construction. Composition of container morphisms is defined in terms of composition of ordinary functions, hence unital and associative by function extensionality. For n-truncated containers F, G, the type of morphisms $F \to G$ is in general not an h-set, but an n-truncated type. As such, Cont is a wild-, not a 1-category. ⌟

Truncated containers form full subcategories of Cont. We write (n,k)-Cont for the wild category of (n,k)-truncated containers. Note that Cont is *not* univalent in the 1-categorical sense: The canonical map taking identifications of containers $F = G$ to categorical isomorphisms $F \cong_{\mathsf{Cont}} G$ in the sense of definition 1.5.6 is *not* an equivalence, unless shapes and positions of the involved containers are sets. In this case, $(0,0)$-Cont is the univalent 1-category of set-truncated containers.

When comparing containers, we are instead going to use the following notion of "sameness" of containers:

Definition 4.1.2. A morphism $(f,u) : F \to G$ is an *equivalence* of containers if both $f : F_{\mathsf{Sh}} \to G_{\mathsf{Sh}}$ and $u(s) : G_{\mathsf{Ps}}(f(s)) \to P_{\mathsf{Ps}}(s)$ for all $s : F_{\mathsf{Sh}}$ are equivalences of types. We write $F \simeq G$ for the type of equivalences of containers. ⌟

While we cannot prove it internally, we think of the wild category of containers as an $(\infty,1)$-category in which $F \simeq G$ is the "correct" notion of weak equivalence, representing the ∞-groupoid of paths $F = G$. By univalence, equivalences of containers correspond exactly to paths between containers. More precisely, equivalences of containers form an *identity system* in the sense of [Uni13, § 5.8]:

Proposition 4.1.3. The family $\_ \simeq \_ : \mathsf{Cont} \to \mathsf{Cont} \to \mathsf{Type}$ is an identity system.

[1] For an overview and some further references, consider [Buc19, §3.3].

Proof. By [Uni13, Theorem 5.8.4.(v)], it suffices to show that $\sum_{G:\mathsf{Cont}} F \simeq G$ is contractible. But by two applications of univalence, we can prove

$$\begin{aligned}(F \simeq G) &\simeq \sum_{u:F_{\mathsf{Sh}} \simeq G_{\mathsf{Sh}}} \prod_{s:F_{\mathsf{Sh}}} G_{\mathsf{Ps}}(u(s)) \simeq F_{\mathsf{Ps}}(s) \\ &\simeq \sum_{p:F_{\mathsf{Sh}} = G_{\mathsf{Sh}}} \prod_{s:F_{\mathsf{Sh}}} G_{\mathsf{Ps}}(p_* s) \simeq F_{\mathsf{Ps}}(s) \\ &\simeq \sum_{p:F_{\mathsf{Sh}} = G_{\mathsf{Sh}}} G_{\mathsf{Ps}} =_p F_{\mathsf{Ps}} \\ &\simeq (F = G)\end{aligned}$$

Therefore, $\sum_G F \simeq G$ is equivalent to $\sum_G F = G$ which is a type of singletons, hence contractible. □

Consequently, equivalences of containers enjoy a J-like rule, that is, the following induction principle:

Problem 4.1.4 (Equivalence induction)**.** Let $G : \mathsf{Cont}$ and $P : \prod_{F:\mathsf{Cont}} F \simeq G \to \mathsf{Type}$. Define a map

$$\mathsf{J}_{\mathsf{Cont}} : P(G, \mathsf{id}_G) \to \prod_{F:\mathsf{Cont}} \prod_{e:F \simeq G} P(F, e)$$

such that $\mathsf{J}_{\mathsf{Cont}}(p, G, \mathsf{id}_G) = p$ for all $p : P(G, \mathsf{id}_G)$. □

This J-rule for containers is especially useful when reasoning about types of fillers of commutative diagrams, i.e. certain types of identifications $f_0; \dots; f_n = g_0; \dots g_k$ of container morphisms.

Lemma 4.1.5. Let $F, G, H : \mathsf{Cont}$ and $e : F \simeq G$. The precomposition map $e; \_ : (G \to H) \to (F \to H)$ is an equivalence of types of container morphisms. In particular, $(e; h = e; h') \simeq (h = h')$ for all $h, h' : G \to H$.

Proof. By application of $\mathsf{J}_{\mathsf{Cont}}$ to $P(F, e) :\equiv \forall H.\, \mathsf{isEquiv}(\lambda h : G \to H.\, e; h)$ it suffices to show that $\mathsf{id}_G; \_$ is an equivalence, which follows from left-unitality, $\mathsf{id}_G; h = h$. □

When constructing a morphism $f : F \to G$, we will oftentimes factor it through (equivalent) auxiliary containers

$$\begin{array}{ccc} F & \xrightarrow{f} & G \\ {\scriptstyle\simeq}\downarrow & & \uparrow{\scriptstyle\simeq} \\ F' & \xrightarrow[f']{} & G' \end{array}$$

This lets us separate the bureaucracy of bringing F and G into a comparable shape from the act of defining an interesting morphism $f' : F' \to G'$. In particular, f is an equivalence of containers if and only if f' is, which is often easier to characterize.

Recall from definition 1.6.3 that the extension of a container is a polynomial endofunctor on Type. If restricted to h-sets, this defines a full and faithful embedding of the category $(0,0)$-Cont into the functor category [hSet, hSet] [AAG05, Theorem. 3.4]. This embedding reflects much of the structure of set-endofunctors back into containers, allowing concise proofs for a number of properties. However, it seems impossible to prove that $[\![\_]\!]$ embeds into such functors (even at the level of objects), unless one is able to deal with an infinite tower of coherences, as we will discuss later in chapter 5. Instead, we aim to perform our constructions directly on containers, without making reference to their extensions as Type-polynomials.

4.2 Indexed containers

Before we define and reason about an operation $F \mapsto \mu F$ constructing fixpoints, we have to recall the definition of indexed containers. Intuitively, fixpoints are specified with respect to a *signature*, binding a number of variables. One of these variables is a "recursion variable", indicating a placeholder for the recursive occurrence of the type. To encode such types as containers, we have to introduce containers whose positions are *indexed* by some arbitrary type representing a number of bound variables:

Definition 4.2.1 (Indexed containers, [Alt+15, § 4])**.** Let $I : \mathsf{Type}$. The type of *I-indexed containers* is $\mathsf{Cont}_I :\equiv \sum_{S:\mathsf{Type}} (I \to S \to \mathsf{Type})$. ⌟

Ordinary containers correspond to unary containers, Cont_1. Each index $i : I$ corresponds to a variable in the type that a container describes. The extent of an indexed container takes I-indexed families of types to types,

$$[\![S \lhd P]\!] : (I \to \mathsf{Type}) \to \mathsf{Type}$$
$$[\![S \lhd P]\!](\vec{X}) :\equiv \sum_{s:S} \prod_{i:I} P_i(s) \to \vec{X}_i$$

We call $(I+1)$-indexed containers *signature containers*; their fixpoints describe inductive types.

Example 4.2.2. In the informal specification $\mathsf{List}(X) = \mu Y.\, 1 + X \times Y$, the expression $L(X,Y) = 1 + X \times Y$ is transcribed into a binary container L, indexed by the type 2. Its positions at index $\mathsf{y} :\equiv \mathsf{inr}(\bullet) : 2$ encode occurrences of the recursion variable

Y, those at $\mathsf{x} := \mathsf{inl}(\bullet) : 2$ occurrences of X. It has two shapes, nil and cons, over which there are positions

$$L_{\mathsf{Ps}}(\mathsf{nil}, \mathsf{x}) := 0 \qquad L_{\mathsf{Ps}}(\mathsf{cons}, \mathsf{x}) := 1$$
$$L_{\mathsf{Ps}}(\mathsf{nil}, \mathsf{y}) := 0 \qquad L_{\mathsf{Ps}}(\mathsf{cons}, \mathsf{y}) := 1 \quad ⌟$$

Morphisms of indexed containers are defined by indexed maps between indexed positions:

Definition 4.2.3. The type of *morphisms* of indexed containers $F, G : \mathsf{Cont}_I$ is

$$F \to G := \sum_{f : F_{\mathsf{Sh}} \to G_{\mathsf{Sh}}} \prod_{i:I, s:F_{\mathsf{Sh}}} G_{\mathsf{Ps}}(i, f(s)) \to F_{\mathsf{Ps}}(i, s) \quad ⌟$$

Definition 4.2.4 (Wild category of indexed containers)**.** I-indexed containers and their morphisms form a wild category Cont_I. The wild subcategory of (n,k)-truncated containers is denoted by $(n,k)\text{-}\mathsf{Cont}_I$. ⌟

In addition to sums and products (figure 4.1), indexed containers are closed under a number of constructions. For any $A : \mathsf{Type}$, the constant indexed container is $\mathsf{k}(A) := (A \lhd \lambda i\, a.\, 0) : \mathsf{Cont}_I$. Each container in I variables is also one in $I+1$ variables: For $F : \mathsf{Cont}_I$, denote by $\langle F \rangle^{\uparrow} : \mathsf{Cont}_{I+1}$ the inclusion given by

$$\langle F \rangle^{\uparrow}_{\mathsf{Ps}}(\mathsf{just}\, i) := F_{\mathsf{Ps}}(i), \quad \langle F \rangle^{\uparrow}_{\mathsf{Ps}}(\mathsf{nothing}) := 0$$

To aid readability, we denote an $(I+1)$-indexed container $(S \lhd \bar{P})$ by $(S \lhd \vec{P}, P)$, where $\vec{P}_i := \bar{P}_{\mathsf{just}(i)}$ and $P := \bar{P}_{\mathsf{nothing}}$. In particular, $\langle S \lhd P \rangle^{\uparrow} \doteq (S \lhd P, 0)$. If we similarly denote $(I+1)$-indexed families by $(\vec{X}, X) : (I+1) \to \mathsf{Type}$, then

$$[\![\langle S \lhd P \rangle^{\uparrow}]\!](\vec{X}, 0) \simeq \sum_{s:S} \prod_{i:I} P_i(s) \to \vec{X}_i \simeq [\![S \lhd P]\!](\vec{X})$$

For $i : I$, the ith projection container is $\pi_i := (1 \lhd \lambda j\, \_.\, i = j) : \mathsf{Cont}_I$. The extent of π_i is exactly the i-th projection from an I-indexed family, namely

$$[\![\pi_i]\!](\vec{X}) \simeq \left(\textstyle\prod_{j:I} (i = j) \to X_j\right) \simeq \vec{X}_i$$

Substitution generalizes to an operation that places a container inside the positions at index $\mathsf{nothing} : I+1$ of an $(I+1)$-indexed container:

Definition 4.2.5. Substitution is an operation $\_[\_] : \mathsf{Cont}_{I+1} \to \mathsf{Cont}_I \to \mathsf{Cont}_I$, defined as follows:

$$(S \lhd \vec{P}, P)[(T \lhd Q)]_{\mathsf{Sh}} := \sum_{s:S} P(s) \to T$$
$$(S \lhd \vec{P}, P)[(T \lhd Q)]_{\mathsf{Ps}} := \lambda i\, (s, f).\, \vec{P}_i(s) + \sum_{p:P(s)} Q_i(f\, p) \quad ⌟$$

$$(F \otimes G)_{\mathsf{Sh}} \coloneqq S \times T \qquad (F \otimes G)_{\mathsf{Ps}} \coloneqq \lambda i\,(s,t).\, P_i(s) + Q_i(t)$$

$$(F \oplus G)_{\mathsf{Sh}} \coloneqq S + T \qquad (F \oplus G)_{\mathsf{Ps}} \coloneqq \lambda i \begin{cases} \mathsf{inl}(s).\, P_i(s) \\ \mathsf{inr}(t).\, Q_i(t) \end{cases}$$

Figure 4.1: Sum and products of I-indexed containers $F \doteq (S \lhd P)$ and $G \doteq (T \lhd Q)$. Shapes are identical to the non-indexed case (definition 1.6.5); positions are additionally indexed by $i : I$.

Substitution into a fixed container behaves functorially:

Problem 4.2.6. Given $F : \mathsf{Cont}_{I+1}$, extend $F[\_]$ to a wild endofunctor on Cont_I.

Construction. Let $F \doteq (S \lhd \vec{P}, P)$, and $(T \lhd Q), (U \lhd R) : \mathsf{Cont}_I$. Substitution acts on a morphism $u : (T \lhd Q) \to (U \lhd R)$ as follows: The shape map has type $\sum_{s:S}(P_s \to T) \to \sum_{s:S}(P_s \to U)$, and is given by postcomposition with $u_{\mathsf{Sh}} : T \to U$ in the second component. Given $s : S$ and $f : P_s \to T$, the position map has type

$$\vec{P}_i(s) + \sum_{p:P(s)} Q_i(u_{\mathsf{Sh}}(f(p))) \to \vec{P}_i(s) + \sum_{p:P(s)} R_i(f(p))$$

and is given by an application of $u_{\mathsf{Ps}}(i, f(p)) : Q_i(u_{\mathsf{Sh}}(f(p))) \to R_i(f(p))$. Preservation of identities and composition follows directly by extensionality of container morphisms. ⌟

It is exactly this endofunctor on indexed container whose algebras we use to describe container fixpoints.

4.3 Least fixpoints of containers

Just like in the non-indexed case, The extension of set-truncated indexed containers defines subcategory of functors $[\mathsf{hSet}^I, \mathsf{hSet}]$ [AAG05, Theorem. 3.4]. In particular, given a signature container $F : \mathsf{Cont}_{I+1}$ and $\vec{X} : I \to \mathsf{hSet}$, the endofunctor $[\![F]\!](\vec{X}, \_) : [\mathsf{hSet}, \mathsf{hSet}]$ admits an initial algebra. This fixpoint can be reflected back into containers, implying that they are closed under least fixpoint, and thus an adequate model of strictly-positive, inductive types.

In [Alt+15], Altenkirch et al. give an explicit construction of this fixpoint. They notice that substitution $F[\_]$ for a fixed *indexed* container F defines an endofunctor of containers, and that this functor admits an initial algebra. The carrier of this algebra, μF, represents the inductive type given by the "signature

container" F. This construction is carried out under the assumption of UIP, i.e. that all types are h-sets. Recently, Damato et al. have expanded upon this idea in a univalent setting [DAL25]. They show, without assuming UIP, that fixpoints of indexed containers exist, at least on a level of container *functors*: for any signature container $F : \mathsf{Cont}_{I+1}$ and $\vec{X} : I \to \mathsf{Type}$, the wild functor $[\![F]\!](\vec{X}, \_) : [\mathsf{Type}, \mathsf{Type}]$ admits an initial algebra whose carrier is given by $[\![\mu F]\!]\vec{X}$. Ideally, one would like to show that this construction is natural in $\vec{X}$ and pull its properties back along $[\![\_]\!]$, thus proving that μF itself is defined universally among *containers*, without needing to reference their interpretations. This, however, is hampered by the fact that we cannot (yet) internally prove that $[\![\_]\!]$ is a suitable embedding of containers and their morphisms, as discussed earlier.

Instead, we retrace the construction of [Alt+15]. We first ensure, for any $F : \mathsf{Cont}_I$, that $F[\_]$-algebras form a wild category (which is not the case for arbitrary wild endofunctors). Secondly, we construct and initial object in this wild category of algebras. This construction does not make any reference to the associated container functors, avoiding the intricacies of $[\![\_]\!]$ as a functor of $(\infty, 1)$-categories.

Problem 4.3.1 ($F[\_]$-algebras)**.** Given $F : \mathsf{Cont}_{I+1}$, define the structure of a wild category $\mathsf{Alg}_{F[\_]}$, such that

- objects are of type $\sum_{G:\mathsf{Cont}_I} F[G] \to G$,
- morphisms $\mathsf{Alg}_{F[\_]}((G,g),(H,h))$ are container morphisms $f : G \to H$ together with a path $f_\square : f \circ g = h \circ F[f]$ ensuring that the square

$$\begin{array}{ccc} F[G] & \xrightarrow{F[f]} & F[H] \\ {\scriptstyle g}\downarrow & f_\square & \downarrow{\scriptstyle h} \\ G & \xrightarrow[f]{} & H \end{array}$$

  commutes.

Construction. Identities and composite morphisms are defined as expected. However, unlike in 1-categories, unitality and associativity of composition is not automatically inherited from the base category Cont_I, since we have to ensure that there are higher paths between composite squares, such as

$$\begin{array}{ccccc} F[G] & \xrightarrow{F[f]} & F[H] & \xrightarrow{F[\mathsf{id}]} & F[H] \\ {\scriptstyle h}\downarrow & f_\square & \downarrow{\scriptstyle g} & \mathsf{id}_\square & \downarrow{\scriptstyle g} \\ G & \xrightarrow[f]{} & H & \xrightarrow[\mathsf{id}]{} & H \end{array} \quad = \quad \begin{array}{ccc} F[G] & \xrightarrow{F[f]} & F[H] \\ {\scriptstyle h}\downarrow & f_\square & \downarrow{\scriptstyle g} \\ G & \xrightarrow[f]{} & H \end{array}$$

Fortunately, these higher paths are straightforward to construct by extensionality of container morphisms. ⌟

We define least fixpoints of containers as initial objects (definition 1.5.4):

Definition 4.3.2. A least fixpoint of a signature container $F : \mathsf{Cont}_{I+1}$ is an initial object in the wild category of $F[\_]$-algebras. ⌟

Spelled out, a least F-fixpoint consists of a *carrier* $M : \mathsf{Cont}_I$ and a structure map $m : F[M] \multimap M$ such that for any other algebra (G, g), the type of $F[\_]$-algebra morphisms $(M, m) \to (G, g)$ is contractible. The latter means exactly that there is a unique container morphism $r : M \to G$ that commutes with the structure maps m and g. As a consequence, the carriers of least fixpoints are equivalent (hence equal) as containers.

We now show, assuming access to W-types, that all containers admit least fixpoints in this sense (theorem 4.3.8). Given $A : \mathsf{Type}$ and a family $B : A \to \mathsf{Type}$, each $w : \mathsf{W}(A, B)$ is a well-founded tree with A-labeled nodes, and $B(a)$-branching subtrees. A path to some node inside w can be defined as an inductive family over $\mathsf{W}(A, B)$:

Definition 4.3.3. *Paths* into $\mathsf{W}(A, B)$, labelled by some $C : A \to \mathsf{Type}$ are a family of types $\bar{\mathsf{W}}_{A,B,C} : \mathsf{W}(A, B) \to \mathsf{Type}$, given inductively by

$$\frac{c : C(a)}{\mathsf{top}(c) : \bar{\mathsf{W}}_{A,B,C}(\mathsf{sup}(a, f))} \qquad \text{and} \qquad \frac{b : B(a) \quad p : \bar{\mathsf{W}}_{A,B,C}(f(b))}{\mathsf{below}(b, p) : \bar{\mathsf{W}}_{A,B,C}(\mathsf{sup}(a, f))}$$

for all $a : A$ and $f : B(a) \to \mathsf{W}(A, B)$. ⌟

Both W and $\bar{\mathsf{W}}$ can be described by unfolding them by one level — the constructors define equivalences

$$\mathsf{W}\text{-in} : \textstyle\sum_{a:A}(B(a) \to \mathsf{W}(A, B)) \xrightarrow{\simeq} \mathsf{W}(A, B)$$

$$\bar{\mathsf{W}}\text{-in}_{a,f} : C(a) + \textstyle\sum_{b:B(a)} \bar{\mathsf{W}}_{A,B,C}(f(b)) \xrightarrow{\simeq} \bar{\mathsf{W}}_{A,B,C}(\mathsf{sup}(a, f))$$

for all $a : A$ and $f : B(a) \to \mathsf{W}(A, B)$.

The container μF represents a type of F-branching trees, thus we use W and $\bar{\mathsf{W}}$ to define its shapes and positions, respectively.

Definition 4.3.4. Let $F \doteq (S \lhd \vec{P}, P) : \mathsf{Cont}_{I+1}$, define $\mu F :\equiv (S^\mu \lhd P^\mu) : \mathsf{Cont}_I$:

$$S^\mu :\equiv \mathsf{W}(S, P) \qquad \text{and} \qquad P_i^\mu :\equiv \bar{\mathsf{W}}_{S,P,\vec{P}_i}$$ ⌟

These let us derive a fixpoint to substitution:

Problem 4.3.5. Define an equivalence of containers $\mathsf{in}_F : F[\mu F] \simeq \mu F$.

Construction. Let $F \doteq (S \lhd \vec{P}, P)$. Shapes of $F[\mu F]$ are $\sum_{s:S}(P(s) \to \mathsf{W}(S,P))$, hence W-in establishes an equivalence with the shapes of μF. Similarly, the positions of $F[\mu F]$ are equivalent to those of μF via $\bar{\mathsf{W}}$-in. ⌟

Together, μF and in_F form an $F[\_]$-algebra. We define a non-dependent recursion principle for maps out of μF, which yields a morphism into *any* other algebra:

Problem 4.3.6 (μ-recursion)**.** Define a recursion principle for morphisms out of μ-containers. That is, for any $F : \mathsf{Cont}_{I+1}$, $G : \mathsf{Cont}_I$ and $\alpha : F[G] \to G$, give a morphism

$$\mathsf{rec}_F(\alpha) : \mu F \to G$$

together with a chosen filler

$$\begin{array}{ccc} F[\mu F] & \xrightarrow{F[\mathsf{rec}_F(\alpha)]} & F[G] \\ {\scriptstyle \mathsf{in}_F}\downarrow & & \downarrow{\scriptstyle \alpha} \\ \mu F & \xrightarrow[\mathsf{rec}_F(\alpha)]{} & G \end{array} \tag{4.1}$$

Construction. Let $F \doteq (S \lhd \vec{P}, P)$ and $G \doteq (T \lhd Q)$. Abbreviate $\mathsf{rec}_F(\alpha)$ by rec. The shape map of α has type $\sum_{s:S}(P(s) \to T) \to T$, so on shapes, rec is defined by W-recursion,

$$\begin{aligned} &\mathsf{rec}_{\mathsf{Sh}} : \mathsf{W}(S,P) \to T \\ &\mathsf{rec}_{\mathsf{Sh}} \coloneqq \mathsf{W}\text{-}\mathsf{rec}(\alpha_{\mathsf{Sh}}) \end{aligned}$$

On positions we define $u : \prod_{w:\mathsf{W}(S,P)} Q_i(\mathsf{rec}_{\mathsf{Sh}}(w)) \to \bar{\mathsf{W}}_{S,P,\vec{P}}(w)$ by W-induction. If $w \doteq \mathsf{sup}(s,f)$, then $u(w)$ is the composite

$$\begin{array}{ccc} Q_i(\alpha_{\mathsf{Sh}}(s, \mathsf{rec}_{\mathsf{Sh}} \circ f)) & \xrightarrow{u(w)} & \bar{\mathsf{W}}_{S,P,\vec{P}}(\mathsf{sup}(s,f)) \\ {\scriptstyle \alpha_{\mathsf{Ps}}(i, \mathsf{rec}_{\mathsf{Sh}} \circ f)}\downarrow & & \uparrow{\scriptstyle \bar{\mathsf{W}}\text{-}\mathsf{in}} \\ \vec{P}_i(s) + \sum_{p \in P(s)} Q_i(\mathsf{rec}_{\mathsf{Sh}}(f(p))) & \xrightarrow[u^*]{} & \vec{P}_i(s) + \sum_{p \in P(s)} \bar{\mathsf{W}}_{S,P,\vec{P}}(\mathsf{sup}(s,f)) \end{array} \tag{4.2}$$

in which u^* applies $u(f(p))$ recursively in the second component of the Σ-type.

To show that (4.1) commutes, it suffices to find a filler for

$$\begin{array}{ccc} F[\mu F] & \xrightarrow{F[\mathsf{rec}_F(\alpha)]} & F[G] \\ {\scriptstyle \mathsf{out}_F}\uparrow & & \downarrow{\scriptstyle \alpha} \\ \mu F & \xrightarrow[\mathsf{rec}_F(\alpha)]{} & G \end{array} \tag{4.3}$$

as in_F is an equivalence with inverse out_F. By extensionality of container morphisms, this follows directly by a single W-induction on the shapes of μF. ⌟

In fact, this morphism is uniquely determined, hence we get a fixpoint for any signature container. We establish this by proving that $\mathsf{rec}_F(\alpha)$ is unique as a coalgebra-to-algebra morphism:

Proposition 4.3.7. For all $F : \mathsf{Cont}_{I+1}$, $G : \mathsf{Cont}_I$ and $\alpha : F[G] \to G$, the type

$$\sum_{\alpha^* : \mu F \to G} \alpha^* = \mathsf{out}_F ; F[\alpha^*] ; \alpha$$

is contractible with center $(\mathsf{rec}_F(\alpha), \mathsf{rec}_F(\alpha)_\square)$, where $\mathsf{rec}_F(\alpha)_\square$ is the filler of diagram 4.3.

Proof. Let $\alpha^* : \mu F \to G$ and $\alpha^*_\square : \alpha^* = \mathsf{out}_F ; F[\alpha^*] ; \alpha$. We need to give a path $p : \mathsf{rec}_F(\alpha) = \alpha^*$, and over it a dependent path $p_\square : \mathsf{rec}_F(\alpha)_\square =_p \alpha^*_\square$ connecting the commuting squares

$$\begin{array}{ccc} F[\mu F] & \xrightarrow{F[\mathsf{rec}_F(\alpha)]} & F[G] \\ {\scriptstyle \mathsf{out}_F}\uparrow & & \downarrow{\scriptstyle \alpha} \\ \mu F & \xrightarrow[\mathsf{rec}_F(\alpha)]{} & G \end{array} \qquad \text{and} \qquad \begin{array}{ccc} F[\mu F] & \xrightarrow{F[\alpha^*]} & F[G] \\ {\scriptstyle \mathsf{out}_F}\uparrow & & \downarrow{\scriptstyle \alpha} \\ \mu F & \xrightarrow[\alpha^*]{} & G \end{array}$$

Let us sketch the proof. By extensionality of morphisms (lemma 1.6.6), the path p can be given as a μF_{Sh}-indexed family of paths. But μF_{Sh} is a W-type, hence we define p by W-induction as a composite of two paths, p_1 and p_2, using the path $\alpha^*_\square$. The composite of these two paths is unique up to a unique dependent path, and from this we build the filler connecting the two squares above. In our mechanized proof,[2] this construction makes extensive use Cubical Agda's primitive support for dependent path types and cube filling operations. This allows us to leave `transport` hell, and instead settle in `PathP` purgatory. □

From this, uniqueness as an algebra follows immediately, assuming that the relevant commutative squares have equivalent types of fillers. Unlike for set-truncated containers (where commutativity of both squares is a propositions) these fillers are data, and we have to ensure that we preserve their structure.

Theorem 4.3.8. For all $F : \mathsf{Cont}_{I+1}$, the pair $(\mu F, \mathsf{in}_F)$ is an initial $F[\_]$-algebra, hence a least fixpoint of F.

[2] `μ-rec-unique'` of [IIIa, `Derivative.Indexed.Mu`]

Proof. Let (G, α) be a second $F[\_]$-algebra. The claim follows from proposition 4.3.7 if the respective types of square-fillers are equivalent. We have

$$(\alpha^* = \mathsf{out}_F ; F[\alpha^*]; \alpha) \overset{(?)}{\simeq} (\mathsf{in}_F ; \alpha^* = \mathsf{in}_F ; \mathsf{out}_F ; F[\alpha^*]; \alpha) \simeq (\mathsf{in}_F ; \alpha^* = F[\alpha^*]; \alpha)$$

in which (?) is an application of lemma 4.1.5: precomposition with in_F is an equivalence of types $\mu F \to G$ and $F[\mu F] \to G$. □

With this we have shown that the construction of least fixpoints of containers à la [Alt+15] is valid in a univalent setting where UIP is unavailable. We succeeded because we aligned our proof obligations with the only primitive at hand: μF is defined in terms of W-types, so any property has to ultimately be established by W-induction. Instead of proving initiality directly like [Alt+15, Proposition 5.2] does, we go through proposition 4.3.7 to show that rec is unique as a coalgebra-to-algebra morphism. Since all proof obligations (namely p and $p_\square$ therein) are ultimately W-indexed families, this can be shown by W-induction alone, without any contentious transport that might require applications of UIP. Assuming UIP, the proof of theorem 4.3.8 becomes trivial: the filler-types are inhabited propositions, so trivially equivalent. Our proof depends on univalence in the form of lemma 4.1.5, and ultimately proposition 4.1.3: equivalence of containers is an identity system.

Introducing a cut in this argument may seem inconsequential, but we believe that it nicely separates the computational aspect from the specification — on one hand rec is a unique fold-like operation (proposition 4.3.7), on the other μF is the minimal solution to a problem (theorem 4.3.8).

In the future, we would like to apply the dual argument to greatest $F[\_]$-fixpoints, where unfolding (corec) gives rise to a terminal coalgebra on νF. The shapes of this fixpoint are an M-*type*, whose existence is derivable both in a setting with UIP [Alt+15], and assuming univalence [ACS15]. In particular, applications of W-induction in proposition 4.3.7 should translate into applications of the coinduction proof principle of M-types. This would give us a unique algebra-to-coalgebra morphism, which would yield a greatest $F[\_]$-fixpoint.

4.4 Removing points from higher types

The goal of this section is to identify, for an arbitrary type, a subtype of its points that can be removed in a well-behaved manner. In the classical world of sets, removal of elements is no big deal: a set is nothing but a collection of discrete things, and discarding individual elements is done without second thought. Types, however, model more that just discrete collections: in Univalent Foundations, one takes the perspective that types model spaces in which

individual points are connected by potentially complicated spaces of paths. All operations we define have to have a continuous interpretation, so in particular, given some $A : \mathsf{Type}$ and a point $a_0 : A$, the type $A \setminus a_0 \coloneqq \sum_{a:A} a_0 \neq a$ embeds into A by removing the entire connected component surrounding a_0, never just the individual point itself.

If A is an h-set, then the connected component around a_0 is contractible. Thus, homotopically, the complement of the embedding $A \setminus a_0 \hookrightarrow A$ is just a point, resembling the classical case. For higher types, we recall the notion of a_0 being an *isolated point* of A, which is sufficient to prove that the component of a_0 is contractible.

Our goal is to characterize the isolated points of a number of inductive types (§ 4.4.1), and to show that doing so for Σ-types is constructively difficult. In § 4.4.2 we demonstrate that they behave just as one would naively expect when removed from a type. Lastly, in § 4.4.3, we give an induction-like principle characterizing functions out of types with a chosen isolated point: If $a : A$ is isolated, then $(A \to B) \simeq (A \setminus a \to B) \times B$ — any function $f : A \to B$ is determined exactly by its restriction to $A \setminus a$ (the points of A not equal to a) and its value $f(a) : B$. This property is in general false for non-isolated points in types of higher truncation level, since the decomposition discards the behaviour of f on the higher path spaces around a.

To demonstrate properties of isolated points, we freely use univalence and its consequence, function extensionality. We believe that many properties also hold under more fine-grained assumptions.

4.4.1 Isolated points

Recall that a type is *discrete* if equality of any pair of its points is decidable. In a univalent world, this is a rather strong property — by Hedberg's Theorem, any such type forms a (homotopy) set, and can thus not have any interesting path structure. In some cases a type might have this property locally, however. Consider for example the sum $A + 1$ for an arbitrary type A. Even if equality in this type is not decidable without further assumptions on A, it should at least be possible to decide $x = \mathsf{inr}(\bullet)$, no matter which $x : A + 1$ we are given. After all, the constructors inl and inr are injective, so case-analysis should yield a decision procedure, even without looking at points of A. We call such points *isolated*:

Definition 4.4.1. A point $a : A$ is *isolated* if $a = b$ is decidable for all $b : A$. That is, isolated points satisfy the predicate

$$\begin{aligned} &\mathsf{isIsolated} : A \to \mathsf{Type} \\ &\mathsf{isIsolated}(a) \coloneqq \textstyle\prod_{b:A} \mathsf{Dec}(a = b) \end{aligned}$$

We denote by $A^\circ \coloneqq \sum_{a:A} \mathsf{isIsolated}(a)$ the subtype of isolated points. ⌟

In this section, we are going to study some properties of isolated points. We are particularly interested to see how they interact with the various type formers such as binary- and dependent sums.

In general, a decision procedure of equality is extra structure on a type; after all, there are possibly many ways to affirm that, *yes*, $a = b$ in an arbitrary type A. Therefore, the type $\mathsf{isIsolated}(a)$ might, *a priori*, not be a proposition. Take for example the circle S^1: there are $\mathbb{Z}$-many proofs of $\mathsf{Dec}(\mathsf{base} =_{S^1} \mathsf{base})$. However, since equality has to be decidable uniformly for a point to be isolated, this does not happen: isolated points *do* have trivial path spaces.[3] To prove this, Kraus et al. give the following "local" version of Hedberg's Theorem:

Lemma 4.4.2 ([Kra+17, Theorem 3.12])**.** If $a : A$ is isolated, then $a = b$ is a proposition for all $b : A$.

Importantly, this shows that being isolated is a property of a point:

Corollary 4.4.3. If $a : A$ is isolated, then $\mathsf{Dec}(a = b)$ is a proposition for all $b : A$.

Proof. The type $\mathsf{Dec}(P)$ is a proposition for any proposition P. So by lemma 4.4.2, $\mathsf{Dec}(a = b)$ is a proposition. □

Proposition 4.4.4. For all $a : A$, $\mathsf{isIsolated}(a)$ is a proposition. In particular, A° embeds as a subtype of A.

Proof. Any type P is a proposition if $P \to \mathsf{isProp}(P)$. To show that $\mathsf{isIsolated}\, a$ is a proposition for any $a : A$, it thus suffices to prove

$$\mathsf{isIsolated}(a) \to \mathsf{isProp}(\mathsf{isIsolated}(a))$$

But $\mathsf{isIsolated}(a) \doteq \forall b.\, \mathsf{Dec}(a = b)$, hence it is a proposition by corollary 4.4.3. □

Taking these together, we see that the subtype of isolated points carves out a discrete part of a type:

Proposition 4.4.5 (Isolated points form a set)**.** For any type A, its subtype of isolated points A° is discrete, hence a set.

Proof. By proposition 4.4.4, $A^\circ \hookrightarrow A$, so given $(a, h_a), (b, h_b) : A^\circ$, it suffices to decide the equality $a = b$ in A. We can do so either via $h_a(b) : \mathsf{Dec}(a = b)$ or $h_b(a) : \mathsf{Dec}(b = a)$. □

[3] In fact S^1 is *perfect*, i.e. has no isolated points at all.

Arbitrary functions do not necessarily reflect isolated points: given $f : A \to B$, a point $a : A$ might not be isolated even if its image $f(a) : B$ is. However, maps which behave "nicely" on path spaces reflect isolated points. This applies in particular to embeddings, which are equivalences on path spaces.

Proposition 4.4.6 (Embeddings reflect isolated points)**.** Let $f : A \hookrightarrow B$ and $a : A$. If $f(a)$ is isolated in B, then a is isolated in A.

Proof. For all $a' : A$, we need to decide $a = a'$. By assumption, we can decide whether $f(a) = f(a')$ or not. If $f(a) \neq f(a')$, then necessarily $a \neq a'$. If $f(a) = f(a')$, we get $a = a'$ since f is an embedding, which we can cancel. □

Consequently, equivalences both preserve and reflect isolated points:

Corollary 4.4.7. If $e : A \simeq B$, then $a : A$ is isolated if and only if $e(a) : B$ is isolated. We write $e^\circ : A^\circ \simeq B^\circ$ for the induced equivalence.

Proof. Both $e : A \to B$ and its inverse $e^{-1} : B \to A$ are embeddings. □

In principle, we could weaken the assumptions of the previous proposition to a function $f : A \to B$ for which *some* $\prod_{x,y} f(x) = f(y) \to x = y$ exists — not necessarily an inverse to cong_f: the proof works no matter which path we pick, and *post hoc* this choice of path is unique, since it originates from an isolated point. In the remainder, however, we will apply proposition 4.4.6 exclusively to embeddings. We can use it, for example, to show that the canonical embeddings $\mathsf{inl} : A \hookrightarrow A + B$ and $\mathsf{inr} : B \hookrightarrow A + B$ both reflect and create isolated points:

Proposition 4.4.8. Let $A, B : \mathsf{Type}$. A point $a : A$ is isolated if and only if $\mathsf{inl}(a) : A + B$ is isolated; similarly for $b : B$ and $\mathsf{inr}(b) : A + B$.

Proof. Let $a : A$. In the forward direction, assume that a is isolated. We need to show that for any $x : A + B$, the type $\mathsf{inl}\, a = x$ is decidable. Consider the case of $x \doteq \mathsf{inl}\, a'$. We know that inl is an embedding, and as such there is an equivalence of path spaces $(\mathsf{inl}\, a = \mathsf{inl}\, a') \simeq (a = a')$. But $(a = a')$ is decidable by assumption, hence $(\mathsf{inl}\, a = \mathsf{inl}\, a')$ is decidable. In case $x \doteq \mathsf{inr}\, b$, the type $\mathsf{inl}\, a = \mathsf{inr}\, b$ is empty, hence decidable. For the converse, apply proposition 4.4.6: The map inl is an embedding, and as such reflects isolated points. □

The above tells us that isolated points distribute over binary sums:

Problem 4.4.9. Construct an equivalence $(A + B)^\circ \simeq A^\circ + B^\circ$.

Construction. Define the obvious forward- and backward maps by case analysis, and prove that points are isolated using proposition 4.4.8. That these maps are mutually inverse follows since being isolated is a proposition (proposition 4.4.4). ⌟

From this we immediately see that $\mathsf{nothing} := \mathsf{inr}(\bullet) : A+1$ is an isolated point, since $\bullet : 1$ is trivially isolated:

Corollary 4.4.10. The point $\mathsf{nothing} : A+1$ is isolated for any type A, and there is an equivalence $(A+1)^\circ \simeq A^\circ + 1$.

This yields the decision procedure for $\prod_{x:A+1} \mathsf{Dec}(x = \mathsf{nothing})$ that we described informally at the beginning of this section.

While it may seem obvious that the isolated points of a binary sum are a sum of isolated points, describing the isolated points of Σ-types is a more subtle affair. First, observe that any dependent pair of isolated points defines an isolated point in the corresponding Σ-type:

Proposition 4.4.11. Let $A : \mathsf{Type}$ and $B : A \to \mathsf{Type}$ with points $a_0 : A$ and $b_0 : B(a_0)$. If both a_0 and b_0 are isolated, then (a_0, b_0) is isolated in $\sum_{a:A} B(a)$.

Proof. Let $a : A$, $b : B(a)$. Assuming that both are isolated, our goal is to decide whether $(a_0, b_0) = (a, b)$ holds or not. By extensionality of path types of dependent sums it suffices to decide the equivalent type $\sum_{p:a_0=a} b_0 = p_*^{-1}(b)$. If $a_0 \neq a$, then this type is empty. Otherwise, we have some $p : a_0 = a$, and the type is inhabited or empty depending on whether $b_0 = p_*^{-1}(b)$ holds or not. □

Definition 4.4.12. For all $A : \mathsf{Type}$ and $B : A \to \mathsf{Type}$, define

$$\Sigma\text{-}\mathsf{isolate}_{A,B} : \sum_{a:A^\circ} B(a)^\circ \to \Big(\sum_{a:A} B(a)\Big)^\circ$$
$$\Sigma\text{-}\mathsf{isolate}_{A,B}\Big((a, h_a), (b, h_b)\Big) := ((a, b), h_{a,b})$$

with $h_{a,b} : \mathsf{isIsolated}((a, b))$ given by proposition 4.4.11. ⌟

Naïvely, we would assume that Σ-isolate is an equivalence, and that isolated points of a type of dependent pairs are exactly pairs of isolated points. Inspecting the fibers of Σ-isolate, we notice that they are all propositions, hence Σ-isolate must be at least an embedding:

Lemma 4.4.13. For all $A : \mathsf{Type}$, $B : A \to \mathsf{Type}$, and $y \doteq ((a, b), \_) : (\Sigma_A B)^\circ$, there is an equivalence $\mathsf{fiber}_{\Sigma\text{-}\mathsf{isolate}}(y) \simeq \mathsf{isIsolated}(a) \times \mathsf{isIsolated}(b)$.

Proof. By extensionality of paths in Σ-types, we rephrase the type of fibers over y in terms of singletons over a and b; there is an equivalence

$$\mathsf{fiber}_{\Sigma\text{-}\mathsf{isolate}}(y) \simeq \sum_{(a',p):\mathsf{singl}(a)} \sum_{(b',\_):\mathsf{singl}(p_* b)} \mathsf{isIsolated}(a') \times \mathsf{isIsolated}(b')$$

The claim follows by contracting away these singletons. □

Proposition 4.4.14. $\Sigma\text{-}\mathsf{isolate}_{A,B}$ is an embedding for all $A : \mathsf{Type}$, $B : A \to \mathsf{Type}$.

Proof. By the previous result, all fibers of $\Sigma\text{-isolate}_{A,B}$ are propositions. □

A converse to proposition 4.4.11 would suffice to prove that Σ-isolate is a surjection, ensuring that it is an equivalence. It turns out that this requirement is not only sufficient, but also a necessary assumption which fully characterizes isolated points in Σ-types.

Theorem 4.4.15. Let $A : \mathsf{Type}$ and $B : A \to \mathsf{Type}$. The following are equivalent:

1. $\Sigma\text{-isolate}_{A,B}$ is an equivalence.
2. $\Sigma\text{-isolate}_{A,B}$ is a surjection.
3. For all $a : A$ and $b : B(a)$, if (a, b) is isolated in $\Sigma_A B$, then both a and b are isolated.

Proof. Since Σ-isolate is an embedding, 4.4.15.(1) and 4.4.15.(2) are clearly equivalent. We show that 4.4.15.(2) if and only if 4.4.15.(3): By lemma 4.4.13, the fiber over an arbitrary $y \doteq ((a, b), h) : (\Sigma_A B)^\circ$ is equivalent to $\mathsf{isIsolated}(a) \times \mathsf{isIsolated}(b)$. Thus, all fibers of Σ-isolate are inhabited if and only of 4.4.15.(3) holds. □

Sums of discrete types over a discrete base are themselves discrete, and in this case Σ-isolate is trivially invertible:

Corollary 4.4.16. If $A : \mathsf{Type}$ is discrete, and $B : A \to \mathsf{Type}$ is a family of discrete types, then $\Sigma\text{-isolate}_{A,B}$ is an equivalence.

Proof. Condition theorem 4.4.15.(3) is vacuously satisfied for discrete types. □

Less trivially, we can state this locally for an isolated point in the base, and obtain a generalization of proposition 4.4.8 from binary- to arbitrary sums:

Proposition 4.4.17. Let $A : \mathsf{Type}$, $B : A \to \mathsf{Type}$, and $a : A^\circ$. Then any $b : B(a)$ is isolated if and only if $(a, b) : \Sigma_A B$ is isolated.

Proof. Only the backward direction is non-trivial, and follows from proposition 4.4.6: since a is isolated, $\lambda b.(a, b)$ is an embedding, and therefore reflects isolated points. □

In general, however, it seems difficult to describe exactly when isolated points distribute this way. In extreme cases, both A and B can be complicated types, whereas their sum $\Sigma_A B$ is entirely trivial. This applies in particular to the type of singletons, $\mathsf{singl}(a_0) :\equiv \sum_{a:A} a_0 = a$.

Proposition 4.4.18. For all types A, the following are equivalent:

1. A is discrete.
2. For all $a_0 : A$, the map

$$\Sigma\text{-isolate}_{A,a_0=\_} : \sum_{a:A^\circ} (a_0 = a)^\circ \to \mathsf{singl}(a_0)^\circ$$

   is an equivalence.

Proof. If A is discrete, then so are its path types, hence $\Sigma\text{-isolate}_{A,a_0=\_}$ is an equivalence by corollary 4.4.16. In the other direction, we can show that every $a_0 : A$ is isolated: Since $\mathsf{singl}(a_0)$ is a contractible type, its center (a_0, refl) is isolated. Thus, by theorem 4.4.15.(3), the first component a_0 must be isolated. □

This lets us describe whether a type is discrete purely in terms of Σ-isolate, which will be useful in situations in which we have control over the types indexing Σ-isolate.

4.4.2 Removing isolated points

Given two types, removing a point from their sum is the same as removing it from either side, and then taking the sum:

Problem 4.4.19. Given $A, B : \mathsf{Type}$, define equivalences

$$(A + B) \setminus \mathsf{inl}(a_0) \simeq (A \setminus a_0) + B$$
$$(A + B) \setminus \mathsf{inr}(b_0) \simeq A + (B \setminus b_0)$$

for all points $a_0 : A$ and $b_0 : B$, respectively.

Construction. Define the obvious maps in either direction by case-analysis. These preserve inequalities since inl and inr are embeddings, and are inverses of each other as inequalities are always propositions. ⌟

Consequently, freely adding a point to a type and then removing it again defines an equivalence of types:

Problem 4.4.20. For all $A : \mathsf{Type}$, define an equivalence $(A + 1) \setminus \mathsf{nothing} \simeq A$.

Construction. The type $(1 \setminus \bullet)$ is empty, so by problem 4.4.19, $(A + 1) \setminus \mathsf{nothing} \simeq A + (1 \setminus \bullet) \simeq A$. ⌟

What if we consider the converse problem? That is, *first* removing a point $a_0 : A$, *then* adding it back into the type. Does this yield a type equivalent to A? In one direction, there is an obvious map

$$\begin{aligned} &\mathsf{replace}_{a_0} : (A \setminus a_0) + 1 \to A \\ &\mathsf{replace}_{a_0}(\mathsf{just}\,(a, \_)) :\equiv a \\ &\mathsf{replace}_{a_0}(\mathsf{nothing}) :\equiv a_0 \end{aligned} \tag{4.4}$$

Classically, this map is an isomorphism of sets whose inverse maps a_0 to $\bullet$. In a univalent setting however, this is problematic. First, it is in general not possible to decide whether some $a : A$ is equal to a_0. Second, this replaces the higher path spaces of a_0 with the trivial ones of $\bullet$.

Example 4.4.21. Consider the circle with its point base : S^1. The circle is connected, hence $\|\mathsf{base} = x\|_{-1}$ for any $x : S^1$. It follows that $S^1 \setminus \mathsf{base} \simeq 0$. So replacing base results in $(S^1 \setminus \mathsf{base}) + 1 \simeq 0 + 1 \simeq 1$, which is evidently not the circle! Removing base removes its entire connected component, and replaces it with a contractible type.

This argument applies to any pointed, connected type (A, a_0). ⌟

Assuming a_0 to be isolated in equation (4.4) alleviates not only the first problem, but also the second — by lemma 4.4.2, $a_0 = a_0$ is contractible, and thus there is no nontrivial path space to be replaced. Indeed, $\mathsf{replace}_{a_0}$ exhibits its classical behavior exactly when a_0 is an isolated point:

Proposition 4.4.22. Let $a_0 : A$. The map $\mathsf{replace}_{a_0} : (A \setminus a_0) + 1 \to A$ is an equivalence if and only if a_0 is isolated in A.

Proof. First, assume that a_0 is isolated. We give an inverse $g : A \to (A \setminus a_0) + 1$ as follows. Define $\mathsf{r} : \prod_{a:A} \mathsf{Dec}\,(a_0 = a) \to (A \setminus a_0) + 1$ by

$$\mathsf{r}(a, d) :\equiv \begin{cases} \mathsf{nothing} & \text{if } d \doteq \mathsf{yes}(\_ : a_0 = a) \\ \mathsf{just}\,(a, h) & \text{if } d \doteq \mathsf{no}(h : a_0 \neq a) \end{cases}$$

For $i_0 : \mathsf{isIsolated}(a_0)$, let $g(a) :\equiv \mathsf{r}(a, i_0(a))$. By corollary 4.4.3, $\mathsf{Dec}(a_0 = a)$ (the type of $i_0(a)$) is a proposition for all $a : A$; we use this to ensure that g computes correctly:

$$\begin{aligned} g(a_0) &\doteq \mathsf{r}(a_0, i_0(a_0)) = \mathsf{r}(a_0, \mathsf{yes}(\mathsf{refl})) \doteq \mathsf{nothing} \\ g(a) &\doteq \mathsf{r}(a, i_0(a)) = \mathsf{r}(a, \mathsf{no}(h)) \doteq \mathsf{just}\,(a, h) \quad \text{where } h : a_0 \neq a \end{aligned}$$

Hence $\mathsf{replace}_{a_0} \circ g = \mathsf{id}$ and $g \circ \mathsf{replace}_{a_0} = \mathsf{id}$.

For the converse, assume that $\mathsf{replace}_{a_0}$ is an equivalence. By corollary 4.4.10, nothing is isolated in $(A \setminus a_0) + 1$, which is preserved by $\mathsf{replace}_{a_0}$ (corollary 4.4.7), hence $\mathsf{replace}_{a_0}(\mathsf{nothing}) \doteq a_0$ is isolated as well. □

Later on, it will become necessary to understand how to remove points from Σ-types. If we think of an A-indexed sum $\sum_{a:A} B(a)$ as a generalization of binary sums $B_0 + B_1$, then we expect removal to behave similarly: Removing some (a, b) should remove $b : B(a)$ from the a-th summand, leaving all other summands unchanged. This is indeed the case, as long as we put some restrictions[4] on the paths of the indexing type A:

Problem 4.4.23. Let $A : \mathsf{Type}$ and $B : A \to \mathsf{Type}$ with points $a_0 : A$ and $b_0 : B(a_0)$, and assume $p : \mathsf{isProp}(a_0 = a_0)$. There is a map

$$\Sigma\text{-}\mathsf{remove}_p : \Big(\sum_{a:A\setminus a_0} B(a)\Big) + \Big(B(a_0) \setminus b_0\Big) \to \Big(\sum_{a:A} B(a)\Big) \setminus (a_0, b_0).$$

Construction. We define $\Sigma\text{-}\mathsf{remove}_p(x)$ by cases. Let

$$\Sigma\text{-}\mathsf{remove}_p(\mathsf{inl}(a, h_a, b)) :\equiv ((a, b), h'_a),$$

where $h'_a : (a_0, b_0) = (a, b) \xrightarrow{\mathsf{cong}_{\mathsf{fst}}} a_0 = a \xrightarrow{h_a} \bot$. In the other case, let

$$\Sigma\text{-}\mathsf{remove}_p(\mathsf{inr}(b, h_b)) :\equiv ((a_0, b), h'_b),$$

and show $h'_b : (a_0, b_0) \neq (a_0, b)$ as follows: Assume to the contrary that there is some $p_b : (a_0, b_0) = (a_0, b)$. From this we obtain (dependent) paths $p'_b : a_0 = a_0$ and $p''_b : b_0 =_{p'_b} b$. Since $a_0 = a_0$ is a proposition, we know that $p'_b = \mathsf{refl}$, hence $b_0 = b$. This is contradictory since we are given $h_b : b_0 \neq b$. ⌟

This map is an equivalence whenever we can decide if we are removing from a chosen index a_0:

Proposition 4.4.24. Let $A : \mathsf{Type}$, $B : A \to \mathsf{Type}$ with $a_0 : A$ and $b_0 : B(a_0)$. If a_0 is an isolated point of A, then Σ-remove of problem 4.4.23 is an equivalence.

Proof. First note that $a_0 = a_0$ is a proposition by lemma 4.4.2, thus the map is well-defined. We construct an inverse

$$\Sigma\text{-}\mathsf{remove}^{-1} : \Big(\sum_{a:A} B(a)\Big) \setminus (a_0, b_0) \to \Big(\sum_{a:A\setminus a_0} B(a)\Big) + \Big(B(a_0) \setminus b_0\Big)$$

[4] In [Ec10, UF.Sets], a_0 is called *h-isolated* if $a_0 = a_0$ is a proposition.

as follows: Introduce $a : A$, $b : B(a)$ and $h : (a_0, b_0) \neq (a, b)$, then decide whether $a_0 = a$ or not. If $p : a_0 = a$, we map to $\mathsf{inr}(p_*(b), h')$, where $h' : b_0 \neq p_*(b)$, which we conclude from h and p. In case that $h : a_0 \neq a$ we map to $\mathsf{inl}((a, h), b)$ directly. It is straightforward to verify that these maps are inverses of each other. □

4.4.3 Grafting

Precomposing a function $f : A \to B$ with the embedding $\mathsf{fst} : A \setminus a_0 \hookrightarrow A$ restricts it to a map $A \setminus a_0 \to B$. This defines an operation

$$\mathsf{detach} : \prod_{a_0 : A} (A \to B) \to (A \setminus a_0 \to B) \times B$$

given by $\mathsf{detach}_{a_0}(f) \coloneqq (f \circ \mathsf{fst}, f(a_0))$. In order to derive the chain rule for derivatives, Abbott et al. [Abb+05] define the inverse of this operation for *discrete* A: Given a function $g : A \setminus a_0 \to B$ and some $b_0 : B$ they call the process of extending g to a map $A \to B$ *grafting*.

We obtain a good notion of grafting for higher types by localizing the assumption of decidable equality: For arbitrary types A, B and $a_0 : A$, detach_{a_0} is an equivalence if a_0 is isolated in A. This defines an induction-like principle for types with a chosen isolated point $a_0 : A^\circ$: Functions $A \to B$ are uniquely determined by a map $A \setminus a_0 \to B$ and a single point $b_0 : B$.

First, let us define *grafting* with respect to an isolated point:

Problem 4.4.25. For types A and B, construct a function

$$\mathsf{graft} : \prod_{a_0 : A^\circ} \Big((A \setminus a_0 \to B) \times B \Big) \to (A \to B).$$

Following [Abb+05], write $[b_0|f]_{a_0} \coloneqq \mathsf{graft}_{a_0}(f, b_0)$ or simply $[b_0|f]$ if $a_0 : A^\circ$ is understood from context. Ensure that computes as follows: for all $a_0 : A^\circ$, $f : A \setminus a_0 \to B$ and $b_0 : B$,

$$[b_0|f]_{a_0}(a_0) = b_0 \qquad \text{and} \qquad \prod_{a : A} \prod_{h : a_0 \neq a} [b_0|f]_{a_0}(a) = f(a, h).$$

Construction. Let $a_0 : A^\circ$, $f : A \setminus a_0 \to B$ and $b_0 : B$. Decide equality with a_0 to define $\mathsf{graft}_{a_0}(f, b_0) : A \to B$ as follows:

$$\mathsf{graft}_{a_0}(f, b_0) \coloneqq \lambda a. \begin{cases} f(a, h) & \text{if } (h : a_0 \neq a) \\ b_0 & \text{otherwise} \end{cases}$$

The first computation rule is straightforward, the second follows from an application of corollary 4.4.3. ⌟

It follows that grafting is an inverse to detach:

Proposition 4.4.26 (graft-induction)**.** Let $A, B : \mathsf{Type}$ and $a_0 : A$. If a_0 is isolated in A, then detach_{a_0} is an equivalence, typed

$$(A \setminus a_0 \to B) \times B \simeq (A \to B).$$

Its inverse is given by $\mathsf{graft}_{a_0} : (A \setminus a_0 \to B) \times B \to (A \to B)$.

Proof. The computation rules of problem 4.4.25 ensure that detach_{a_0} and graft_{a_0} are mutually inverse. □

As presented here, graft characterizes non-dependent functions out of types with an isolated point $a_0 : A$. This generalizes to dependent functions, in that we can derive an equivalence $\left((\prod_{a:A\setminus a_0} B(a)) \times B(a_0)\right) \simeq \left(\prod_{a:A} B(a)\right)$ for families B over A. We understand this as an "elimination operation" in the sense of [MM04, §4]: we can define arbitrary sections $f : \prod_{a:A} B(a)$ of the *motive* B by supplying two *methods* $f^* : \prod_{a:A\setminus a_0} B(a)$ and $b_0 : B(a_0)$. The extra structure of $a_0 : A^\circ$ lets us, in a sense, "pattern match" on $a = a_0$ and $a \neq a_0$.

4.5 Derivatives of containers

The derivative of a container G represents a type of G-shaped trees in which a chosen subtree has been removed. For traditional containers, this can be implemented as an operation $G \mapsto \partial G$ that removes a chosen position over each shape [Abb+05]: Given $G \doteq (T \lhd Q)$, define ∂G to have as shapes pairs $(t, q) : \sum_T Q$, over which the positions are $Q_t \setminus q$. To ensure that ∂ is well-behaved, it is characterized by a universal property in the full subcategory of cartesian morphisms:

Definition 4.5.1 (cartesian morphism)**.** Let $F \doteq (S \lhd P)$ and $G \doteq (T \lhd Q)$. A container morphism $f : F \to G$ is *cartesian* if $f_{\mathsf{Ps}}(s) : Q(f_{\mathsf{Sh}}(s)) \to P(s)$ is an equivalence of types for each $s : S$. We denote the subtype of cartesian morphisms by

$$F \multimap G :\equiv \sum_{f:S\to T} \prod_{s:S} Q_{f(s)} \simeq P_s$$

The forgetful map defines a (subtype) embedding $(F \multimap G) \hookrightarrow (F \to G)$ which we will leave implicit in the remainder.

Any equivalence of containers is in particular a cartesian morphism. In this chapter we will thus denote equivalence by $F \circ\!\!-\!\!\circ G$ instead of $F \simeq G$. ⌟

Cartesian morphisms capture the intuition of a "linear" map between datatypes: if the map of positions of $f : F \to G$ is an equivalence, then its extension as a natural transformation $[\![f]\!] : [\![F]\!] \Rightarrow [\![G]\!]$ can only permute data, not drop or duplicate it. Understanding cartesian morphisms as linear maps lets us specify ∂G by a universal *mapping-in* property: Abbott et al. prove that if G has discrete positions, then $F \multimap \partial G$ is in 1-to-1 correspondence with $F \otimes \mathsf{Id} \multimap G$. In a cartesian morphism $(f, u) : F \otimes \mathsf{Id} \multimap G$, the position maps are equivalences $u_s : G_{\mathsf{Ps}}(f(s)) \simeq F_{\mathsf{Ps}}(s) + 1$. In particular, there is a position $u_s^{-1}(\mathsf{inr}(\bullet)) : G_{\mathsf{Ps}}(fs)$ in the output which does not correspond to a position in the input, $F_{\mathsf{Ps}}(s)$, indicating the hole. The derivative ∂G computes a container that represents a type of "one-hole contexts of G", giving categorical semantics to McBride's derivative of regular types [McB01].

As we have seen in § 4.4.2, removing points from a type is a subtle process in a univalent setting: a position $q : G_{\mathsf{Ps}}(t)$ is not simply a discrete point, but comes with a potentially complicated type of paths around it. If we wanted to encode morphisms into ∂G by the same universal property, we would have to avoid the entire connected component around q, that is, find some type of "hole" $H(q)$ such that $G_{\mathsf{Ps}}(fs) \simeq F_{\mathsf{Ps}}(s) + H(q)$. But this type now depends on q and its higher path structure, and can no longer be expressed uniformly as a simple product with the fixed container Id.

From this, we can devise two ways forward: Either we characterize the derivative in terms of a more fine-grained universal property that takes the dependency on higher paths into account, or we only take derivatives with respect to positions whose path types are more uniform. We take the second approach, and define a derivative in terms of *isolated* positions:

Definition 4.5.2. The derivative of a container $\partial(S \lhd P) :\equiv (S' \lhd P')$ has shapes $S' :\equiv \sum_{s:S} P_s^\circ$ and positions $P'(s, p) :\equiv P_s \setminus p$. ⌟

As originally conceived, derivatives of containers are only universal with respect to discrete containers:

Definition 4.5.3. A container $(S \lhd P)$ is *discrete* if $P(s)$ is discrete for all $s : S$. ⌟

For discrete containers, our definition agrees with the original definition:

Proposition 4.5.4. If F is a discrete container, then so is ∂F. If shapes of F are in addition a set, then ∂F coincides with the traditional derivative as per [Abb+05].

Proof. Let $F \doteq (S \lhd P) : \mathsf{Cont}$. For all $s : S$ and $p : P_s^\circ$, $\partial F_{\mathsf{Ps}}(s, p) \doteq P_s \setminus p$ is a subtype of P_s, hence decidable. By assumption P_s is discrete, thus $P_s^\circ \simeq P_s$, hence $F_{\mathsf{Sh}} \simeq \sum_{s:S} P_s$, which matches the traditional definition. □

4.5.1 Basic laws of derivatives

Derivatives of containers earn their name by their interaction with other operations on containers: derivatives of constants are zero, derivatives distribute over sums, and products follow the familiar Leibniz rule. Let us now investigate to which extent our derivative still respects these basic laws.

It is easy to see that derivatives of constants are always zero, and that $\partial\mathsf{Id}$ is the constant $\mathsf{k}(1)$. Both factor through the following observation:

Proposition 4.5.5. Let $S : \mathsf{Type}$ and $P : S \to \mathsf{Prop}$. There is an equivalence of containers

$$\partial(S \lhd P) \multimap (\textstyle\sum_S P \lhd 0)$$

In particular, we have

1. $\partial(\mathsf{Id}) \multimap \mathsf{k}(1)$,
2. $\partial(\mathsf{k}(A)) \multimap \mathsf{k}(0)$ for all $A : \mathsf{Type}$.

Proof. Since P_s is a proposition, we know that $P_s^\circ \simeq P_s$ and $P_s \setminus p \simeq 0$. Thus,

$$\begin{aligned}\partial(S \lhd P) &\multimap (((s,p) : \textstyle\sum_{s:S} P_s^\circ) \lhd P_s \setminus p)\\ &\multimap (((s,p) : \textstyle\sum_{s:S} P_s) \lhd 0)\end{aligned}$$

□

Similarly, we convince ourselves that ∂ distributes over (binary) sums, and that derivatives of products follow a Leibniz rule:

Proposition 4.5.6. For containers F, G, the following hold:

1. Sum rule: $\partial(F \oplus G) \multimap \partial F \oplus \partial G$
2. Leibniz rule: $\partial(F \otimes G) \multimap (\partial F \otimes G) \oplus (F \otimes \partial G)$

Proof. Let $F \doteq (S \lhd P)$ and $G \doteq (T \lhd Q)$. Both equivalences are established like in the discrete setting (cf. [Abb+05, Proposition 6.3 and 6.4]), with one exception: to derive the Leibniz rule, one needs to show that isolated points distribute over binary sums in

$$\sum_{s:S}\sum_{t:T}(P_s + Q_t)^\circ \simeq \sum_{s:S}\sum_{t:T} P_s^\circ + Q_t^\circ,$$

which is done via problem 4.4.9. □

4.5.2 Universal property

Let us now show that this generalized derivative is still universal. We do so by showing that ∂ is a functor on a wild category of containers and cartesian morphisms, and then constructing a wild adjunction $\_ \otimes \mathsf{id} \dashv \partial$. From this, we can deduce a mapping-in property in form of an equivalence $F \multimap \partial G \simeq F \otimes \mathsf{Id} \multimap G$, for *arbitrary* containers F and G.

First, observe that ∂ acts on container morphisms if they are cartesian.

Definition 4.5.7. The derivative of a morphism $(f, u) : F \multimap G$ is

$$\partial(f,u) : \partial F \multimap \partial G$$
$$\partial(f,u) \coloneqq (f', u')$$

It has the following components:

1. On shapes, $f' : \sum_{s:S} P^\circ_s \to \sum_{t:T} Q^\circ_t$ applies f to the first component and $(u_s^{-1})^\circ : P^\circ_s \simeq Q^\circ_{fs}$ to the second.

2. On positions, $u'_{s,p} : G_{fs} \setminus u_s^{-1}(p) \simeq F_s \setminus p$ is obtained from u_s, which respects the removed point p. (cf. corollary 4.4.7). ⌟

That the morphism is cartesian is crucial; to define the $\partial(f)_{\mathsf{Ps}}$, the map f_{Ps} needs to have an inverse.

Cartesian morphism are clearly closed under composition, so let us denote by $\mathsf{Cont}^{\multimap}$ the wide wild subcategory of Cont consisting of all containers, but only cartesian morphisms. On this wild category, ∂ is a functor:

Problem 4.5.8. Define a wild endofunctor $\partial : \mathsf{Cont}^{\multimap} \to \mathsf{Cont}^{\multimap}$.

Construction. Definitions 4.5.2 and 4.5.7 give actions on objects and morphisms, respectively. By an appeal to extensionality of morphisms (lemma 1.6.6), we obtain identifications $\partial(\mathsf{id}_F) = \mathsf{id}_{\partial F}$ and $\partial(fg) = (\partial f)(\partial g)$. ⌟

We can restrict this to an endofunctor on the *entire* 1-category of set-truncated containers, which ensures that all set-truncated containers are differentiable, without having to assume discreteness. We do so by observing that that the derivative of a container does not only preserve discreteness (proposition 4.5.4), but also its truncation level — as long as shapes are at least sets, and positions are at least propositions. This holds because even for a completely arbitrary type, its isolated points always form a set.

Proposition 4.5.9. For $n \geq 0$ and $k \geq -1$, the derivative of an (n,k)-truncated container is (n,k)-truncated.

Proof. Let $(S \lhd P)$ an (n,k)-truncated container. By proposition 4.4.5 P_s° is a 0-truncated type and S is n-truncated, thus $\partial(S \lhd P)_{\mathsf{Sh}} \doteq \sum_{s:S} P_s^\circ$ is n-truncated. Positions are k-truncated since $P_s \setminus p$ embeds into P_s. □

Corollary 4.5.10. Taking derivatives is an endofunctor on the category of set-truncated containers, $\partial : (0,0)\text{-}\mathsf{Cont}^{\multimap} \to (0,0)\text{-}\mathsf{Cont}^{\multimap}$. □

We believe that analogues of this hold for higher truncation levels. Symmetric containers, for example, form a bicategory $(1,0)\text{-}\mathsf{Cont}^{\multimap}$, and it should be straightforward (albeit tedious) to show that ∂ restricts to a pseudofunctor on this bicategory.

We now show that this generalized derivative is right-adjoint to $\_ \otimes \mathsf{Id}$, verifying that it has the desired universal property. When restricted to h-sets this reduces to an ordinary adjunction of 1-categories. We define this adjunction in terms of unit- and counit natural transformations, and discuss how this relates to the original construction of Abbott et al.

Problem 4.5.11. Define a wild adjunction $(\eta, \varepsilon) : \_ \otimes \mathsf{Id} \dashv \partial$, that is:

1. Two families of morphisms
$$\eta : \prod_{F:\mathsf{Cont}} F \multimap \partial(F \otimes \mathsf{Id}) \qquad \text{and} \qquad \varepsilon : \prod_{G:\mathsf{Cont}} \partial G \otimes \mathsf{Id} \multimap G$$
2. For all $f : F \multimap G$, fillers of naturality squares
$$\begin{array}{ccc} F & \xrightarrow{\eta^F} & \partial(F \otimes \mathsf{Id}) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \partial(f \otimes \mathsf{Id})} \\ G & \xrightarrow[\eta^G]{} & \partial(G \otimes \mathsf{Id}) \end{array} \qquad \text{and} \qquad \begin{array}{ccc} \partial F \otimes \mathsf{Id} & \xrightarrow{\varepsilon^F} & F \\ {\scriptstyle \partial f \otimes \mathsf{Id}}\downarrow & & \downarrow{\scriptstyle f} \\ \partial G \otimes \mathsf{Id} & \xrightarrow[\varepsilon^G]{} & G \end{array}$$
3. Fillers for the zigzag-diagrams
$$\begin{array}{ccc} F \otimes \mathsf{Id} & \xrightarrow{\mathsf{id}} & F \otimes \mathsf{Id} \\ & {\scriptstyle \eta^F \otimes \mathsf{Id}} \searrow \quad \nearrow {\scriptstyle \varepsilon^{F \otimes \mathsf{Id}}} & \\ & (\partial(F \otimes \mathsf{Id})) \otimes \mathsf{Id} & \end{array} \qquad \text{and} \qquad \begin{array}{ccc} & \partial(\partial G \otimes \mathsf{Id}) & \\ {\scriptstyle \eta^{\partial G}} \nearrow & & \searrow {\scriptstyle \partial(\varepsilon^G)} \\ \partial G & \xrightarrow{\mathsf{id}} & \partial G \end{array}$$

Construction. Let $F \doteq (S \lhd P)$ and define $\eta^F : F \multimap \partial(F \otimes \mathsf{Id})$. On shapes, let
$$\eta^F_{\mathsf{Sh}} : S \to \textstyle\sum_{(s,\_):S\times 1} (P_s + 1)^\circ$$
$$\eta^F_{\mathsf{Sh}}(s) \coloneqq ((s, \bullet), \mathsf{nothing})$$

where nothing is isolated in $P_s + 1$ by corollary 4.4.10. On positions, define

$$\eta^F_{\mathsf{Ps}} : \prod_{s:S} (P_s + 1) \setminus \mathsf{nothing} \simeq P_s$$

using problem 4.4.20: adding a point and then removing it does not change P_s.

On the other side, let $G \doteq (T \lhd Q)$ and define the counit $\varepsilon^G : (\partial G) \otimes \mathsf{Id} \multimap G$ as follows: The shape map $\varepsilon^G_{\mathsf{Sh}} : \sum_{t:T} Q^\circ_t \times 1 \to T$ is the first projection. On positions, we are given $t : T$ and $q : Q^\circ_t$ and need to provide an equivalence

$$\varepsilon^G_{\mathsf{Ps}}(t, q, \_) : Q_t \simeq (Q_t \setminus q) + 1$$

We can do so thanks to proposition 4.4.22: removing the *isolated* point q and replacing it by $\bullet : 1$ does not change Q_t.

To construct the zigzag-fillers, we apply the necessary extensionality principles for functions, equivalences and sum types. We are left to construct paths that are almost refl; only some proofs of isolation and removal need to be compared up to propositional equality. Construction of the naturality squares for η and ε is done similarly. ⌟

In their original construction, Abbott et al. only establish isomorphisms between hom-sets $F \otimes \mathsf{Id} \multimap G$ and $F \multimap \partial G$, natural in F. This falls short of defining a proper adjunction since ∂G is left undefined for non-discrete containers G. We can however complete the search for a suitable subcategory of differentiable containers [Abb+05, p. 14]: Our derivative is defined functorially for *all* containers (problem 4.5.8), and restricting the above wild adjunction to set-truncated containers yields the following:

Theorem 4.5.12. In the 1-category of set-truncated containers $(0,0)$-$\mathsf{Cont}^{\multimap}$, ∂ is right-adjoint to tensoring $\_ \otimes \mathsf{Id}$. □

From this, we can extract the familiar natural isomorphism of hom-sets in $(0,0)$-$\mathsf{Cont}^{\multimap}$. In fact, the same argument lets us obtain a natural equivalence of hom-types for arbitrary containers, which otherwise would be somewhat tedious to establish: there is an equivalence $(F \multimap \partial G) \simeq (F \otimes \mathsf{Id} \multimap G)$ natural in $F, G : \mathsf{Cont}^{\multimap}$, with underlying map

$$\begin{aligned} \_^\sharp &: (F \multimap \partial G) \to (F \otimes \mathsf{Id} \multimap G) \\ f^\sharp &:= \varepsilon_G \circ (f \otimes \mathsf{Id}) \end{aligned}$$

Interestingly, the proof of [Abb+05, Theorem 5.1] already uses $\_^\sharp$ to show naturality of the hom-set isomorphism in F.

Furthermore, we can iterate the adjunction and easily obtain a notion of n-fold derivatives. Denote by $\mathsf{y}(A) \coloneqq (1 \lhd A)$ the container of "A-tuples". The n-th derivative $\partial^n(G)$ encodes a type of G-terms with n holes, and satisfies a similar universal property:

Corollary 4.5.13 (Iterated derivative)**.** For all $n : \mathbb{N}$, there is an adjunction

$$\_ \otimes \mathsf{y}[n] \dashv \partial^n$$

in the 1-category $(0,0)$-$\mathsf{Cont}^{\multimap}$.

We conjecture that this extends to an adjunction on the entire wild category $\mathsf{Cont}^{\multimap}$, but one has to carefully check that the data of the wild adjunction in problem 4.5.11 composes coherently.

4.5.3 Solutions to differential equations

To some extent we can also solve differential equations involving ∂. In particular, given a container F, we can ask if it has an anti-derivative, i.e. some G for which $\partial G \multimap F$. Interestingly, ∂ has fixed-points, that is containers that are their own derivative.

The prototypical example of such a fixed-point is the container of finite multisets, or *bags*. We have seen in chapter 2 that bags are best expressed with a groupoid of shapes encoding the necessary symmetries. Using their representation as a symmetric container $\mathsf{Bag} \coloneqq (\mathsf{FinSet} \lhd \mathsf{El})$, we can show the following:

Proposition 4.5.14. The bag-container is a fixed-point of derivation: there is an equivalence $\partial\mathsf{Bag} \multimap \mathsf{Bag}$.

Proof. First, note that finite sets are closed under addition and removal of points: if X is finite, then so are $X + 1$ and $X \setminus x$ for all $x : X$. On shapes, we give an equivalence $\sum_{X:\mathsf{FinSet}} \mathsf{El}(X)^\circ \simeq \mathsf{FinSet}$ in terms of mutually inverse functions f and g. From left to right, define $f(X, x_0) \coloneqq X \setminus x_0$; the other way let $g(X) \coloneqq (X + 1, \mathsf{nothing})$. By univalence, finite sets are equal if their carrier types are equivalent, so f and g are inverses by proposition 4.4.22 and problem 4.4.20. Positions over a shape $(X, x_0) : \sum_{X:\mathsf{FinSet}} \mathsf{El}(X)^\circ$ are related by the identity equivalence, that is

$$\mathsf{Bag}_{\mathsf{Ps}}(f(X, x_0)) \doteq \mathsf{El}(f(X, x_0)) \doteq X \setminus x_0 \doteq \partial\mathsf{Bag}_{\mathsf{Ps}}(X, x_0) \qquad \square$$

Unlike in classical analysis, where the exponential function is the unique solution to the differential equation $f' = f$ with initial condition $f(0) = 1$, the situation for containers is more nuanced: While Bag is a solution for $\partial F \multimap F$ such that $F[\mathsf{k}\,0] \multimap \mathsf{k}\,1$, it is far from being the only one. This is not entirely

unexpected: containers are closely related to Joyal's *combinatorial species* (as discussed e.g. in Yorgey's thesis [Yor14, p. 67]), and these are known to have many non-isomorphic solutions even for simple differential equations, as shown by Labelle in [Lab86].

The proof of proposition 4.5.14 applies to any subuniverse of types closed under addition and removal of single points:

Proposition 4.5.15. Let $P : \mathsf{Type} \to \mathsf{Prop}$ a predicate such that for all $A : \mathsf{Type}$, the following hold:

- $P(A) \to P(A + 1)$
- $P(A) \to \prod_{a_0 : A^\circ} P(A \setminus a_0)$

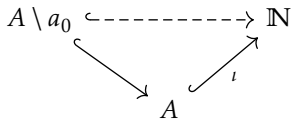
This defines a container $\mathsf{Bag}_P := (\sum_{\mathsf{Type}} P \lhd \mathsf{fst})$ such that $\partial \mathsf{Bag}_P \multimap \mathsf{Bag}_P$. □

For example, we can show that countable collections are a ∂-fixpoint:

Example 4.5.16 (Countable multisets)**.** A type A is *countable* if $P(A) := \|A \hookrightarrow \mathbb{N}\|$ holds. Countable types are closed under addition of points: any $\iota : A \hookrightarrow \mathbb{N}$ extends to

$$\begin{aligned} \iota^+ &: (A + 1) \to \mathbb{N} \\ \iota^+(\mathsf{just}\, a) &:= \iota(a) + 1 \\ \iota^+(\mathsf{nothing}) &:= 0 \end{aligned}$$

which is easily shown to be an embedding. Similarly, given $a_0 : A^\circ$, we embed



where the left map is the subtype-embedding $A \setminus a_0 \hookrightarrow A$. This defines a large ∂-fixpoint container. ⌟

4.5.4 The chain rule

In addition to the basic properties of the previous sections, we expect the derivative to satisfy an analogue of the chain rule $(f \circ g)' = (f' \circ g) \cdot g'$, which describes how derivatives distribute over composition. In our setting, substitution _[_] takes on the role of composition, and for discrete containers, Abbott et al. show that $\partial F[G]$ is indeed isomorphic to $(\partial F)[G] \otimes \partial G$.

In this section, we show the following: we can construct the underlying map $(\partial F)[G] \otimes \partial G \multimap \partial(F[G])$ and show that it is an embedding of containers (proposition 4.5.23). Assuming that it is an equivalence for arbitrary untruncated containers F and G is, however, inconsistent (theorem 4.5.25), or at least a classical principle if assumed for set-truncated F and G (corollary 4.5.27).

First, let us define the aforementioned map underlying the chain rule. This map has to distribute isolated points over a Σ-type, which is only well-behaved for discrete types, and lets us prove both no-go theorems by an application of proposition 4.4.18.

Problem 4.5.17 (The lax chain rule)**.** For all $F, G : \mathsf{Cont}$, define the *directed* or *lax* chain rule; a morphism

$$\mathsf{chain}_{F,G} : (\partial F)[G] \otimes \partial G \multimap \partial(F[G])$$

Construction. Let $F \doteq (S \lhd P)$ and $G \doteq (T \lhd Q)$. As usual, we have to construct a map on shapes and an equivalence of positions. On shapes, our goal is a map

$$\Big(\sum_{(s,p):\sum_{s:S} P_s^\circ} (P_s \setminus p \to T)\Big) \times \sum_{t:T} Q_t^\circ \to \Big(\sum_{(s,f):\sum_{s:S}(P_s \to T)} \Big(\sum_{p:P_s} Q_{fp}\Big)^\circ\Big)$$

Let us first reshape the left side by some equivalences. By re-associating the sums, we obtain

$$\begin{aligned} &\Big(\sum_{(s,p):\sum_{s:S} P_s^\circ} P_s \setminus p \to T\Big) \times \sum_{t:T} Q_t^\circ \\ \simeq &\sum_{(s,p):\sum_{s:S} P_s^\circ} \sum_{(f,t):(P_s \setminus p \to T) \times T} Q_t^\circ \end{aligned}$$

The induction principle for graft tells us that the types $(P_s \setminus p \to T) \times T$ and $P_s \to T$ are equivalent (proposition 4.4.26), thus we simplify to

$$\simeq \sum_{(s,p):\sum_{s:S} P_s^\circ} \sum_{f:P_s \to T} Q_{fp}^\circ$$

By permuting the sum yet again, we are left with

$$\simeq \sum_{(s,f):\sum_{s:S}(P_s \to T)} \Big(\sum_{p:P_s^\circ} (Q_{fp})^\circ\Big)$$

Denote this equivalence by λ. Now, the left and the right only differ in

$$\sum_{p:P_s^\circ} (Q_{fp})^\circ \qquad \text{vs.} \qquad \Big(\sum_{p:P_s} Q_{fp}\Big)^\circ$$

Definition 4.4.12 gives us a map $\Sigma\text{-isolate}_{P_s,Q_{f(\_)}} : \sum_{p:P_s^\circ}(Q_{fp})^\circ \to \left(\sum_{p:P_s} Q_{fp}\right)^\circ$, hence $\text{chain}^{\text{Sh}}_{F,G} \coloneqq \Sigma(\text{id}, \Sigma\text{-isolate}_{P_s,Q_{f(\_)}}) \circ \lambda$.

To construct the equivalence on positions, let $s : S$, $p_0 : P_s^\circ$, $f : P_s \setminus p_0 \to T$, $t : T$ and $q_0 : Q_t^\circ$. First, we reshape by Σ-remove (which is an equivalence by proposition 4.4.24) to distribute removal of the isolated points p_0 and q_0,

$$\left(\sum_{p:P_s} Q_{[t|f]_{p_0}(p)}\right) \setminus (p_0, q_0) \simeq \left(\sum_{p:P_s\setminus p_0} Q_{[t|f]_{p_0}(p)}\right) + \left(Q_{[t|f]_{p_0}(p_0)} \setminus q_0\right)$$

then apply the graft-computation rules (problem 4.4.25) to $[t|f]_{p_0} : P_s \to T$

$$\simeq \left(\sum_{p:P_s\setminus p_0} Q_{f(p)}\right) + (Q_t \setminus q_0)$$

which defines an equivalence of the desired type. ⌟

Remark 4.5.18. The construction for 4.5.17 essentially factors $\text{chain}_{F,G}$ into

$$\begin{array}{ccc} (\partial F)[G] \otimes \partial G & \xrightarrow{\text{chain}_{F,G}} & \partial(F[G]) \\ \downarrow & & \downarrow \\ H_0 & \xrightarrow[\eta]{} & H_1 \end{array}$$

for some implicitly defined containers H_0 and H_1. The shape map

$$\eta_{\text{Sh}} : \sum_{s:S} \sum_{f:P_s\to T} \sum_{p:P_s^\circ} Q^\circ_{fp} \to \sum_{s:S} \sum_{f:P_s\to T} \left(\sum_{p:P_s} Q_{fp}\right)^\circ$$

then simply applies $\Sigma\text{-isolate}_{P_s,Q_{f(\_)}}$, which is the source of non-invertibility of $\text{chain}_{F,G}$. We will make this factorization explicit in the derivation of the chain rule for indexed containers (problem 4.5.29). ⌟

Before looking for an inverse to chain, let us make the following abbreviations:

Definition 4.5.19. The chain rule is *strong* for F and G if $\text{chain}_{F,G}$ is an equivalence of containers. It is *(globally) strong* if that is the case for *all* containers. ⌟

As such, whether the chain rule is strong only depends on Σ-isolate.

Proposition 4.5.20. Let $F = (S \triangleleft P)$ and $G = (T \triangleleft Q)$. The following are equivalent propositions:

1. $\text{chain}_{F,G}$ is an equivalence of containers.
2. $\Sigma\text{-isolate}_{P_s,Q_{f(\_)}}$ is an equivalence for all $s : S$ and $f : P_s \to T$.

Proof. By inspection of the definition of chain in terms of Σ-isolate. □

The chain rule is strong when restricted to *discrete* containers, which is in line with [Abb+05, Proposition 6.6].

Theorem 4.5.21. For discrete containers F, G, $\mathsf{chain}_{F,G}$ is strong. In particular, it is an isomorphism in the 1-category of set-truncated containers.

Proof. Positions of F and G are discrete, so by corollary 4.4.16 Σ-isolate is an equivalence. By proposition 4.5.20, $\mathsf{chain}_{F,G}$ is an equivalence of containers. □

Note that Σ-isolate is always an embedding (cf. proposition 4.4.14). Therefore, $\mathsf{chain}_{F,G}$ always embeds $(\partial F)[G] \otimes \partial G$ as a sub-container inside $\partial(F[G])$, even if it is not strong:

Definition 4.5.22. A morphism $f : F \multimap G$ is an embedding of containers if $f_{\mathsf{Sh}} : F_{\mathsf{Sh}} \to G_{\mathsf{Sh}}$ is an embedding. ⌟

Proposition 4.5.23. For all $F, G : \mathsf{Cont}$, $\mathsf{chain}_{F,G} : (\partial F)[G] \otimes \partial G \multimap \partial(F[G])$ is an embedding.

Proof. By proposition 4.4.14, Σ-isolate is an embedding. □

This lets us slightly weaken the equivalence of 4.5.20.

Corollary 4.5.24. Let $F = (S \lhd P)$ and $G = (T \lhd Q)$. The following are equivalent propositions:

1. $\mathsf{chain}_{F,G}$ is strong.
2. $\Sigma\text{-}\mathsf{isolate}_{P_s, Q_{f(\_)}}$ is surjective for all $s : S$ and $f : P_s \to T$.

Proof. By the equivalent characterizations given in 4.4.15. □

We have to conclude however, that it is impossible to have a globally strong chain rule, at least in the presence of provably non-discrete types:

Theorem 4.5.25. The following are equivalent propositions:

1. *Every* type is discrete.
2. For all containers F and G, $\mathsf{chain}_{F,G}$ is an equivalence.

Proof. If every type is discrete, then so is every container, hence the chain rule is always an equivalence by theorem 4.5.21. In the other direction, use proposition 4.4.18 to show that any given type A is discrete — that is, given some $a_0 : A$, prove that $\Sigma\text{-}\mathsf{isolate}_{A,a_0=\_}$ is an equivalence. We do so by applying proposition 4.5.20 to containers $F \coloneqq (1 \lhd A)$ and $G \coloneqq ((a : A) \lhd a_0 = a)$. □

The circle S^1 is provably not discrete — if it were, it would be a set. Consequently, globally assuming a strong chain rule is inconsistent in the presence of types of higher truncation level. The proof of theorem 4.5.25 gives us an explicit pair of groupoid-truncated containers for which the chain rule is not strong:

Corollary 4.5.26. At $F \coloneqq (1 \lhd S^1)$ and $G \coloneqq ((x : S^1) \lhd \mathsf{base} = x)$, the chain rule is not strong, i.e. $\neg\,\mathsf{isEquiv}(\mathsf{chain}_{F,G})$. □

If instead we restrict ourselves to the world of sets, then we can conclude that a globally strong chain rule exists if and only if arbitrary equalities are decidable:

Corollary 4.5.27. The following are equivalent:

1. Every set is discrete.
2. In the 1-category of set-truncated containers, $\mathsf{chain}_{F,G}$ is an isomorphism for all pairs of containers F and G.

Proof. In the 1-category $(0,0)$-$\mathsf{Cont}^{\sim}$, the types of equivalences- and isomorphisms of containers are equivalent. The claim follows by inspection of the proof of theorem 4.5.25, and ensuring that the same argument applies even when all types involved are sets. □

This is of course still a very classical assumption, even if it is not inconsistent: In an impredicative setting, where $\mathsf{Prop} \simeq \Omega$ for some small $\Omega : \mathsf{Type}$, either condition implies the law of the excluded middle: Prop is an h-set, so Ω must be as well. If Ω were discrete, $P = \top$ would be decidable for any $P : \mathsf{Prop}$. From one immediately concludes $P + \neg P$.

We have seen that our definition of derivative behaves nicely even in the presence of non-discrete types, in that we retain the ways in which it interacts with sums and products (proposition 4.5.6). Its interaction with substitution, however, is more subtle. While we do obtain a chain rule, it is now *directed* or *lax* (problem 4.5.17), and whether we can strengthen it to an equivalence depends on the pair of containers involved (proposition 4.5.20). Indeed, assuming the latter for any pair of containers is inconsistent in the presence of higher inductive types such as the circle S^1, or equivalent to a non-constructive principle if restricted to sets.

4.5.5 The binary chain rule

We can lift derivatives to indexed containers. For an I-indexed container, we can take a derivative with respect to any index $i : I$, as long as equality with i is decidable:

Definition 4.5.28 (Derivative of I-ary containers)**.** Let $F \doteq (S \lhd P) : \mathsf{Cont}_I$ and $i : I^\circ$. The ith derivative $\partial_i F \coloneqq (S' \lhd P') : \mathsf{Cont}_I$ is defined as follows:

$$S' \coloneqq \sum_{s:S} P_i(s)^\circ$$

$$P'(j,s,p) \coloneqq \begin{cases} P_i(s) \setminus p & \text{if } i = j \\ P_j(s) & \text{otherwise} \end{cases}$$ ⌟

In order to reduce visual clutter, we investigate only derivatives of *binary* containers, indexed by $2 = 1+1$. However, our arguments apply to $(I+1)$-indexed containers in general. In particular, we write ∂_0 and ∂_1 for the two possible derivatives of a binary container, that is $\partial_{\mathsf{inl}(\bullet)}$ and $\partial_{\mathsf{inr}(\bullet)}$. In the unary case, we omit the subscript and simply write ∂ for $\partial_\bullet$.

Derivatives of indexed containers satisfy the same basic laws for constants, sums and products as in the unary case. Derivatives of composites follow a lax chain rule, which now accounts for multiple indices. As promised in remark 4.5.18, its construction factors into smaller steps that make it obvious why, in general, it is not invertible:

Problem 4.5.29 (Lax chain rule for binary containers)**.** Let $F : \mathsf{Cont}_2$ and $G : \mathsf{Cont}_1$. Define a cartesian morphism

$$\mathsf{chain}_{F,G} : \partial_0 F[G] \oplus \Big(\partial_1 F[G] \otimes \partial G\Big) \multimap \partial(F[G])$$

Construction. Let $F \doteq (S \lhd P)$, $G \doteq (T \lhd Q)$. We define auxiliary containers H_1, H_2 and factor the morphism through a pair of equivalences as follows:

$$\begin{array}{ccc} \mathsf{L} \coloneqq \partial_0 F[G] \oplus \Big(\partial_1 F[G] \otimes \partial G\Big) & \overset{\mathsf{chain}_{F,G}}{\multimap} & \partial(F[G]) =: \mathsf{R} \\ \lambda \big\downarrow & & \big\downarrow \rho \\ H_1 & \xrightarrow[\eta]{} & H_2 \end{array}$$

Following the argument in problem 4.5.17 and some type yoga, we see that the type of shapes of L is equivalent to

$$U_1 \coloneqq \sum_{s:S} \sum_{f : P_1(s) \to T} P_0(s)^\circ + \sum_{p : P_1(s)^\circ} Q(fp)^\circ \tag{4.5}$$

Denote this equivalence by $f_1 : \mathsf{L}_{\mathsf{Sh}} \simeq U_1$, and define $H_1 \coloneqq (U_1 \lhd \mathsf{L}_{\mathsf{Ps}} \circ f_1^{-1})$. This defines the equivalence $\lambda : \mathsf{L} \multimap\!\!\circ\, H_1$ to the left.

On the other side we obtain an equivalence $f_2 : U_2 \simeq \mathsf{R}_{\mathsf{Sh}}$ by distributing $(\_)^\circ$ over binary sums:

$$\begin{aligned} U_2 &\coloneqq \sum_{s:S} \sum_{f:P_1(s)\to T} P_0(s)^\circ + \Big(\sum_{p:P_1(s)} Q(fp)\Big)^\circ \\ &\simeq \sum_{s:S} \sum_{f:P_1(s)\to T} \Big(P_0(s) + \sum_{p:P_1(s)} Q(fp)\Big)^\circ \\ &\simeq \mathsf{R}_{\mathsf{Sh}} \end{aligned}$$

Let $H_2 \coloneqq (U_2 \lhd \mathsf{R}_{\mathsf{Ps}} \circ f_2)$, which yields the equivalence $\rho : H_2 \circ\!\!-\!\!\circ \mathsf{R}$ to the right.

Let us now define $\eta : H_1 \multimap H_2$. Its map on shapes has type

$$\eta_{\mathsf{Sh}} : \Big(\sum_{s:S} \sum_{f:P_1(s)\to T} P_0(s)^\circ + \sum_{p:P_1(s)^\circ} Q(fp)^\circ\Big) \to \Big(\sum_{s:S} \sum_{f:P_1(s)\to T} P_0(s)^\circ + \Big(\sum_{p:P_1(s)} Q(fp)\Big)^\circ\Big)$$

As in problem 4.5.17, define η_{Sh} by applying Σ-isolate to the right summand, $\sum_{p:P_1(s)^\circ} Q(fp)^\circ$. On positions, the equivalence $\eta_{\mathsf{Ps}}(u) : H_2^{\mathsf{Ps}}(\eta_{\mathsf{Sh}}(u)) \simeq H_1^{\mathsf{Ps}}(u)$ is defined by cases, depending on which side of the sum $u : U_1$ falls. Let $s : S$, $f : P_1(s) \to T$. In case of $u \doteq (s, f, \mathsf{inl}\, p_0)$ for $p_0 : P_0(s)^\circ$, our goal is to give

$$(P_0(s)^\circ + B) \setminus \mathsf{inl}(p_0) \simeq (P_0(s)^\circ \setminus p_0) + B$$

for $B \coloneqq \sum_{p:P_1(s)} Q(fp)$, which is an instance of problem 4.4.19.

When $u \doteq (s, f, \mathsf{inr}(p_1, q))$ for some $p_1 : P_1(s)^\circ$ and $q : Q(fp)^\circ$, we rewrite the type of positions as follows:

$$\begin{aligned} H_2^{\mathsf{Ps}}(\eta_{\mathsf{Sh}}(s, f, \mathsf{inr}(p_1, q))) &\doteq \Big(P_0(s) + \sum_{p:P_1(s)} Q(fp)\Big) \setminus \mathsf{inr}(p_1, q) \\ &\simeq P_0(s) + \Big(\sum_{p:P_1(s)} Q(fp)\Big) \setminus (p_1, q) \qquad (4.6) \\ &\simeq P_0(s) + \sum_{p:P_1(s)\setminus p} Q(fp) + (Q(f\,p_1) \setminus q) \qquad (4.7) \\ &\doteq H_1^{\mathsf{Ps}}(s, f, \mathsf{inr}\,(p_1, q)) \end{aligned}$$

In (4.6), we move the pair (p_1, q) to the right of the sum (problem 4.4.19). For (4.7), we split the Σ-type by applying proposition 4.4.24. This is justified since both p_1 and q are isolated points, and together, the pair (p_1, q) is isolated in $\sum_{p:P_1(s)} Q(fp)$ by proposition 4.4.11. ⌟

This factorization makes it obvious that the binary chain rule is again an embedding of containers, and strong whenever isolated points distribute over dependent sums:

Proposition 4.5.30. For all $F \doteq (S \lhd P) : \mathsf{Cont}_2$ and $G \doteq (T \lhd Q) : \mathsf{Cont}_1$, the following are equivalent:

1. $\mathsf{chain}_{F,G}$ is an equivalence of unary containers.
2. For all $s : S$ and $f : P_1(s) \to T$, $\Sigma\text{-}\mathsf{isolate}_{P_1(s),Q(f(\_))}$ is an equivalence.

Proof. The morphism $\mathsf{chain}_{F,G}$ is an equivalence if and only if η_{Sh} in the above construction is an equivalence of types. This in turn is exactly the case when the second condition holds. □

Corollary 4.5.31. For $F : \mathsf{Cont}_2$ and $G : \mathsf{Cont}_1$, $\mathsf{chain}_{F,G}$ is an embedding. □

Like in theorem 4.5.21, this is the case for discrete containers:

Proposition 4.5.32. For discrete $F : \mathsf{Cont}_2$ and $G : \mathsf{Cont}_1$, $\mathsf{chain}_{F,G}$ is strong. □

4.5.6 Derivatives of least fixpoints

Let us now characterize derivatives of least fixpoints. Let $F \doteq (S \lhd P_0, P_1) : \mathsf{Cont}_2$. If we think of μF as an F-branching tree, then $\partial \mu F$ represents a type of trees with a hole. Such a hole must be either in the position of a leaf or occur recursively in a branch, which are given by types of positions P_0 and P_1. Hence, the derivative of μF is again tree-shaped, with branching given by $\partial_0 F$ and $\partial_1 F$: If F is discrete, there is an equivalence $\mu\text{-}\mathsf{rule}_F : \mu F' \circ\!\!-\!\!\circ \partial(\mu F)$ in which $F' := \langle \partial_0 F[\mu F] \rangle^{\uparrow} \oplus (\langle \partial_1 F[\mu F] \rangle^{\uparrow} \otimes \pi_1)$.

We derive the μ-rule by *recursion* from the chain rule: $\mu\text{-}\mathsf{rule}_F$ is a morphism out of $\mu F'$, hence uniquely determined by some morphism out of $F'[\partial \mu F]$, given by $\mathsf{chain}_{F,\mu F}$. This lets us show that, for untruncated containers, the μ-rule is in general *lax*, and that its invertibility depends on the invertibility of $\mathsf{chain}_{F,\mu F}$. This approach is quite different from the original proof, but matches the intuition discussed in ([Abb+05, Proposition 6.5]): μF is a fixpoint for substitution, so it must be possible to describe its derivative in terms of the chain rule.

First, observe that μ-recursion (problem 4.3.6) preserves cartesian morphisms.

Proposition 4.5.33. For any $\alpha : F[G] \multimap G$, the morphism $\mathsf{rec}_F\, \alpha : \mu F \to G$ is cartesian.

Proof. Let $F \doteq (S \lhd \vec{P}, P)$, $G \doteq (T \lhd Q)$, and abbreviate $\mathsf{rec}_F(\alpha)$ by rec. Recall from diagram 4.2 that over a shape $w \doteq \mathsf{sup}(s, f) : \mathsf{W}(S, P)$, the map of positions is given

by the composite

$$\begin{array}{ccc} Q_i(\alpha_{\mathsf{Sh}}(s, \mathsf{rec}_{\mathsf{Sh}} \circ f)) & \xrightarrow{u} & \bar{\mathsf{W}}_{S,P,\vec{P}}(\mathsf{sup}(s,f)) \\ {\scriptstyle \alpha_{\mathsf{Ps}}(i,\mathsf{rec}_{\mathsf{Sh}} \circ f)}\downarrow & & \uparrow{\scriptstyle \bar{\mathsf{W}}\text{-in}} \\ \vec{P}_i(s) + \sum_{p \in P(s)} Q_i(\mathsf{rec}_{\mathsf{Sh}}(f(p))) & \xrightarrow[u^*]{} & \vec{P}_i(s) + \sum_{p \in P(s)} \bar{\mathsf{W}}_{S,P,\vec{P}}(\mathsf{sup}(s,f)) \end{array}$$

where u^* is defined recursively in terms of u. From the induction hypothesis it follows that u^* is an equivalence. By assumption $\alpha_{\mathsf{Ps}}(i, \mathsf{rec}_{\mathsf{Sh}} \circ f)$ is an equivalence, hence all maps in this composite are. □

Consequently, any cartesian morphism $F'[G] \multimap G$ induces a uniquely determined morphism of shape $\mu F' \multimap G$, in particular for $G \coloneqq \partial \mu F$:

Problem 4.5.34 (Lax μ-rule)**.** For any binary container $F : \mathsf{Cont}_2$, define

$$\begin{aligned} \mu\text{-rule}_F &: \mu F' \multimap \partial(\mu F) \\ \mu\text{-rule}_F &= \mathsf{rec}_{F'}(\alpha) \end{aligned}$$

where $F' \coloneqq \langle \partial_0 F[\mu F] \rangle^{\uparrow} \oplus (\langle \partial_1 F[\mu F] \rangle^{\uparrow} \otimes \pi_1) : \mathsf{Cont}_2$ and $\alpha : F'[\partial \mu F] \multimap \partial \mu F$ is defined in terms of $\mathsf{chain}_{F,\mu F}$.

Construction. First note that F' is a container in two variables, and substitution into the second replaces π_1, i.e. there is an equivalence

$$\beta_Y : F'[Y] \multimap \partial_0 F[\mu F] \oplus (\partial_1 F[\mu F] \otimes Y)$$

for any choice of $Y : \mathsf{Cont}_1$. In particular, for $Y \doteq \partial(\mu F)$, the codomain of $\beta_{\partial \mu F}$ is exactly the domain of the binary chain rule (cf. problem 4.5.29). Hence, we define α as the following composite:

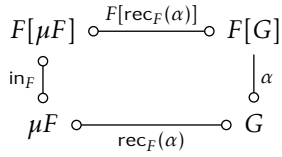
$$\begin{array}{ccc} F'[\partial(\mu F)] & \xrightarrow{\alpha} & \partial(\mu F) \\ {\scriptstyle \beta_{\partial(\mu F)}}\downarrow & & \downarrow{\scriptstyle \partial(\mathsf{in}_F)} \\ \partial_0 F[\mu F] \oplus (\partial_1 F[\mu F] \otimes \partial(\mu F)) & \xrightarrow[\mathsf{chain}_{F,\mu F}]{} & \partial(F[\mu F]) \end{array}$$

Lastly, let $\mu\text{-rule}_F \coloneqq \mathsf{rec}_{F'}(\alpha) : \mu F' \multimap \partial(\mu F)$. ⌟

One advantage of defining the μ-rule by recursion is that it highlights the dependency on the chain-rule. First, observe that recursion reflects equivalence, in the following sense:

Lemma 4.5.35. Let $\alpha : F[G] \multimap G$. If $\mathsf{rec}_F(\alpha)$ is an equivalence, then so is α.

Proof. Substitution $F[\_]$ preserves equivalences, so by 3-for-2 for equivalences of containers, α on the right is an equivalence:



□

Thus, a strong μ-rule for some container necessarily implies a strong chain rule between F and μF:

Proposition 4.5.36. Let $F : \mathsf{Cont}_{I+1}$. If $\mu\text{-rule}_F$ is an equivalence, then so is $\mathsf{chain}_{F,\mu F}$.

Proof. Assume that $\mu\text{-rule}_F$ is an equivalence. In problem 4.5.34, $\mu\text{-rule}_F$ is defined by recursion from some α, hence α is an equivalence by lemma 4.5.35. But α is just $\mathsf{chain}_{F,\mu F}$ wedged between equivalences, so the latter is an equivalence. □

We would like to establish the converse to this as well, but cannot do so directly, as the converse of lemma 4.5.35 does not hold: in general, there are fixed points $\varphi : F[G] \multimap G$ such that μF and G are not equivalent as containers. However, μF is the least among such fixed points, and embeds into any other (pre-)fixed point:

Lemma 4.5.37. If $\alpha : F[G] \multimap G$ is an embedding, then so is $\mathsf{rec}_F(\alpha) : \mu F \multimap G$.

Proof. The shape map $\mathsf{rec}_{F'}(\alpha)_{\mathsf{Sh}} : \mathsf{W}(S,P) \to T$ is given by W-recursion. As such it is an embedding by lemma 1.4.7 □

This tells us that $\mu\text{-rule}_F$ embeds $\mu F'$ as a subcontainer inside of $\partial(\mu F)$:

Lemma 4.5.38. For all $F : \mathsf{Cont}_{I+1}$, the shape map $(\mu\text{-rule}_F)_{\mathsf{Sh}}$ is an embedding.

Proof. By the indexed analogue of proposition 4.5.23, the shape map of the chain rule is an embedding. This lifts to α, and from there to $\mu\text{-rule}_F \doteq \mathsf{rec}_F(\alpha)$ by lemma 4.5.37. □

From this, we can conclude that a strong chain rule alone is enough to prove a strong μ-rule:

Proposition 4.5.39. Let $F : \mathsf{Cont}_{I+1}$. If $\mathsf{chain}_{F,\mu F}$ is an equivalence, then so is $\mu\text{-rule}_F$.

Proof. Abbreviate $\mu^{\mathsf{Sh}} :\equiv (\mu\text{-rule}_F)_{\mathsf{Sh}}$. Since $\mu\text{-rule}_F$ is always an embedding (lemma 4.5.38), it suffices to show that μ^{Sh} is a surjection. In fact, it is a split surjection: all of its fibers are inhabited, not just merely inhabited. Assume $\mathsf{chain}_{F,\mu F}$ to be an equivalence. By proposition 4.5.30, this is equivalent to isolated pairs $(p_1, \bar{w})$ having isolated components $p_1 : P_1(s)$ and $\bar{w} : \bar{\mathsf{W}}(f\, p_1)$, for all $s : S$ and $f : P_1(s) \to \mathsf{W}(S, P_1)$. This property is exactly what is needed to show that all fibers of the shape map are inhabited: we give a term

$$\prod_{(w,\bar{w}):\sum_{w:\mathsf{W}(S,P_1)} \bar{\mathsf{W}}(P_0)^\circ} \mathsf{fiber}_{\mu^{\mathsf{Sh}}}(w, \bar{w})$$

by induction, first on $w : \mathsf{W}(S, P_1)$, then on $\bar{w}$. That is, assume $w \doteq \mathsf{sup}(s, f)$ for $s : S$ and $f : P_1(s) \to \mathsf{W}(S, P_1)$, and that fibers of μ^{Sh} over $f(p_1)$ are inhabited for all $p_1 : P_1(s)$. Now, inspect $\bar{w}$. First, consider the case $\bar{w} \doteq (\mathsf{top}(p_0), h_0)$ for $p_0 : P_0(s)$ and $h_0 : \mathsf{isIsolated}(\mathsf{top}(p_0))$. We conclude that p_0 itself is isolated in $P_0(s)$, and directly construct an inhabitant of the fiber. In the other case, we have $\bar{w} \doteq (\mathsf{below}(p_1, \bar{w}'), h_1)$ for $p_1 : P_1(s)$ and $\bar{w}' : \bar{\mathsf{W}}(f(p_1))$, together with a proof $h_1 : \mathsf{isIsolated}(p_1, \bar{w}')$. From our assumption and h_1 it follows that both p_1 and $\bar{w}'$ are isolated. From this, and the induction hypothesis (fibers over $f(p_1)$ are inhabited), we recursively construct another inhabitant of the fiber. □

Thus, whether the μ-rule is strong depends only on the chain-rule:

Theorem 4.5.40. For any container $F : \mathsf{Cont}_{I+1}$, the following are equivalent:

1. $\mu\text{-rule}_F$ is an equivalence.
2. $\mathsf{chain}_{F,\mu F}$ is an equivalence.

Proof. One direction is proposition 4.5.36, the other is proposition 4.5.39. □

With this at hand, we can give a proof that the μ-rule is strong for discrete containers that factors solely through the properties of the chain rule. First, note that $\bar{\mathsf{W}}$ preserves discreteness:

Lemma 4.5.41. If $B, C : A \to \mathsf{Type}$ are families of discrete types, then $\bar{\mathsf{W}}_{A,B,C}$ is a discrete family.

Proof. By induction on $\mathsf{W}(A, B)$ and unfolding via $\bar{\mathsf{W}}$-in. □

This implies that discrete containers are closed under μ:

Lemma 4.5.42. If $F : \mathsf{Cont}_{I+1}$ is a discrete container, then so is μF. □

Hence, we obtain a strong μ-rule for discrete containers:

Proposition 4.5.43 ([Abb+05, Proposition 8.1])**.** If $F : \mathsf{Cont}_{I+1}$ is discrete, then $\mu\text{-rule}_F$ is an equivalence.

Proof. By theorem 4.5.40, it suffices to show that $\mathsf{chain}_{F,\mu F}$ is an equivalence. But μF is discrete (lemma 4.5.42), and the chain-rule between discrete containers is strong (proposition 4.5.32). □

4.6 Conclusion

We have seen that extending a derivative operation to Type-valued containers is possible: If the obstruction to a general derivative is that positions are not necessarily discrete, simply restrict a derivative to the subtype of those positions that are! This generalization does not come for free, however. Although it is universal and retains basic properties, the chain rule is no longer in general invertible, and neither is the μ-rule. In fact, the chain rule is non-invertible for some higher containers. Assuming that it is invertible for set-truncated containers implies a constructive taboo. Interestingly, the status of the μ-rule is somewhat weaker: In case of the chain rule the contradiction arises because we are free to apply it to an arbitrary pair of containers. In theorem 4.5.40, however, the constraints are tighter — there is one degree of freedom (the container F), and whether the rule is strong depends on that and μF, which is canonically derived from F. We would like to know: Is a strong μ-rule a constructive taboo in the same way that a strong chain rule is?

In their original work [Abb+05], Abbott et al. derive a fixed-point rule also for the largest fixed point, νF. Many of their intermediate lemmata hold for arbitrary fixed points, and the same is true for a number of our results: The construction of the lax chain rule in problem 4.5.17, for example, only depends on the property of $\mathsf{in}_F : F[\mu F] \multimap \mu F$ being *some* $F[\_]$-fixpoint, not necessarily the smallest. We could thus give for any fixpoint $\varphi : F[\varphi F] \multimap\!\!\circ\ \varphi F$ a "φ-rule" as an embedding $\varphi\text{-rule} : \mu F' \multimap \partial(\varphi F)$ in which $F' \coloneqq \langle \partial_0 F[\varphi F]\rangle^{\uparrow} \oplus (\langle \partial_1 F[\varphi F]\rangle^{\uparrow} \otimes \pi_1)$.

In another direction, we would like to describe more than just one-hole contexts. The *linear exponential* $[H, F]$ of containers F and H [Abb+03, Def. 4.1] exists if there is a universal morphism from $\_ \otimes H$ to F. If F is discrete, then $\partial F = [\mathsf{Id}, F]$. The adjunction of theorem 4.5.12 is such a universal morphism, natural in all F, hence $\partial = [\mathsf{Id}, \_]$ as functors. Iterating the adjunction, we can describe n-hole contexts as linear exponentials $\partial^n = [\mathsf{y}[n], \_]$. We conjecture that $[H, \_]$ exists for set-truncated H with finitary positions, since any such H is "just" a big coproduct of representables $\mathsf{y}[n_i]$, This would give us a type of contexts with "H-shaped" holes.

5 Conclusions & Future work

In this thesis we have given some evidence that homotopy type theory is a convenient framework for describing datatypes with symmetries, as long as one embraces the higher structure of types. Even though set-quotients are convenient to use and compute nicely, they truncate the very thing we care about — the symmetries that relate different representations of the same data.

Instead, one should model symmetries by at least considering groupoids. By putting symmetries into the identifications of shapes many constructions and properties of higher containers become just as intuitive as they are for ordinary set-truncated containers. This mirrors the intuition of Kock that in a higher-groupoidal setting, the difference between *polynomial* and *analytic* "evaporates" [Koc12, p. 354]. We believe that our proposed definition of a derivative of Type-valued containers fits this pattern nicely.

Nonetheless, set-based containers are still useful as a specification language: even though action containers so far only model a non-recursive fragment of strictly positive types (products, coproducts and exponentials with constants), they still cover a variety of examples. Types and functions can be specified as action containers and their morphisms, and then be lifted to symmetric containers by means of a well-behaved, 2-categorical translation. Furthermore, the equivalence of action- and skeletal symmetric containers lets us measure some additional strictness-requirements that our restriction to sets imposes. In this setting we profit especially from the insights of *univalent group theory* [Bez+24], as many insights transfer from h-groups to skeletal groupoids.

An interesting conclusion that we can draw from a combination of both chapter 3 and chapter 4 is the following: A container $(S \triangleleft P)$ is "$(\infty, 0)$-truncated" if $S : \mathsf{Type}$ but P is a family of h-sets. Such containers are closed under all operations on containers considered in this thesis, because shapes never appear positively in any of these operations. This applies in particular to the derivative, and we get another variation of corollary 4.5.27: The chain rule is strong for $(\infty, 0)$-truncated containers if and only if all sets have decidable equality. In practice, it is anyway difficult to come up with a non-contrived example of a

type whose positions in a representation as a container are genuine h-groupoids. We believe that $(\infty, 0)$-truncated containers sit at a sweet-spot between fully unrestricted higher- and too-strict set-truncated containers, worthy of further study.

Before we conclude this thesis, we would like to point out some directions that we think are worth pursuing in the future.

Constructive taboos for non-wellfounded trees. Even though set-based notions of non-wellfounded trees are constructively deficient, characterizing the underlying non-constructive obstruction is a worthwhile task of reverse mathematics. As discussed in remark 2.2.22, the functor $\mathsf{U}(X) \coloneqq 1 + X^2/\sim$ exhibits the same behaviour as finite multisets. We conjecture that the proof of theorem 2.2.15 boils down to existence of some *torsion* of a quotient container, i.e. some $g : G_s$ and $n : \mathbb{N}$ such that $g^n = 1$. In the case of U and M this element is swapping of two elements. Of course, whether this is the case in general is undecidable,[1] but it might serve as some demarcation between the constructively possible and impossible.

Furthermore, we would like to weaken the assumptions of theorem 2.3.13: Is it possible to obtain a largest fixpoint of finite multisets from one in groupoids? Whether higher versions of the axiom of choice such as $\mathsf{AC}_{1,0}$ are consistent with univalence is an open question. We would like to know if it is possible to reduce the assumptions again to LLPO.

Cartesian closure of action containers. The category of set-truncated containers is cartesian closed [ALS10], meaning that container capture certain higher-order concepts. Is the same true for action containers? We have characterized action containers as families of group actions, and [AR20, Theorem 2.11] characterizes cartesian closure of $\mathsf{Fam}(\mathcal{C})$ in terms of cartesian multi-closure of the base category $\mathcal{C}$. If cartesian closure of ordinary containers is an example of this ($(0,0)$-Cont is equivalent to $\mathsf{Fam}(\mathsf{hSet}^{\mathsf{op}})$), can we use the same proof for action containers?

Obstacles for substitution-closure Hasegawa shows that analytic functors are closed under composition, and computes an explicit presentation [Has02, Corollary 1.18]. This presentation is the data of an action container with finite positions, and its symmetry groups are given as direct products of *wreath products*, just like in proposition 3.4.21. Translated into the language of containers, their result states that, for any pair of quotient containers $F \doteq (S \triangleleft P \,/\, G)$ and $F' \doteq$

[1] Being a torsion-element of a group is a Markov property, hence undecidable even for finitely presentable groups by the Adian-Rabin theorem [Nyb22].

$(T \triangleleft Q \,/\, H)$ whose positions are finite sets, there exists some $F[F']$ such that $[\![F[F']]\!] = [\![F]\!] \circ [\![F']\!]$. Shapes are $F[F']_{\mathsf{Sh}} = \sum_{s:S} (P_s \to T) \,/\, G_s$ (as expected), and for each $s : S$ and $f : P_s \to T$, there is a group of symmetries

$$F[F']_{\mathsf{Sm}}(s, [f]) = K(s, [f]) = \prod_{t:T} G|_{\mathsf{fiber}_f(t)} \wr H_t \tag{5.1}$$

where $G|_{\mathsf{fiber}_f(t)}$ is a subgroup of G_s. Unfortunately, there are some issues with this construction; it seems like the action of this subgroup identifies too many elements in the quotient.

Example 5.0.1. Consider the containers $F := (1 \triangleleft 4/R)$ and $G := (2 \triangleleft 1/1)$, where the group R is generated by the permutation $r := (12)(43)$ (given in cycle notation). If evaluated at a finite set X, the sets $[\![F[G]]\!](X)$ and $[\![F]\!]([\![G]\!](X))$ are both finite — they are finite sums of quotients of finite types. In particular, by a counting argument, we can compute the cardinality of either set, and conclude that they differ. Therefore, the induced container functors disagree.

The idea is the following: since the symmetries of G are trivial, the wreath product (equation (5.1)) simplifies to a subgroup of R. For each $f : [4] \to [2]$, $K(\bullet, [f])$ is either 1 or R, hence acts on maps of type $[4] \to [2]$ in the extension, i.e. black-and-white colorings of a four-element set. The action is either trivial, or simultaneously swaps two of the for elements. Enumerating all such colorings, one sees that some are identified in $[\![F[G]]\!]([2])$, but not in $[\![F]\!]([\![G]\!]([2]))$. On one hand, counting the elements of the composition of extensions at the two-element set $[2]$, there are two classes, totaling

$$\#([\![F]\!]([\![G]\!]([2]))) = \#([\![F]\!]([2] + [2])) = 16 + 120 = 136 \tag{5.2}$$

elements. On the other, following Hasegawa's definition of $F[G]$, we have

$$[\![F[G]]\!]([2]) \simeq \sum_{y:[4]\to[2]/R} \frac{[4] \to [2]}{K(\bullet, y)}$$

The action of R on $[4] \to [2]$ divides it into 10 orbits, with a total of

$$\#([\![F[G]]\!]([2])) = 9 \cdot 2 + 10 \cdot 2 + 16 \cdot 4 + 16 \cdot 1 + 16 \cdot 1 = 134 \tag{5.3}$$

elements. ⌟

We have attempted to replicate Hasegawa's construction for action containers with finite positions. The family of groups K is defined by induction on a set-quotient, and each group in this family is identified as a subgroup of some fixed ambient group A. The type of subgroups of A is a subtype of embeddings into

the carrier of A, hence must be an h-set. Therefore, the family is well-defined if given $f \sim f'$, we can show that $K(s,[f]) = K(s,[f'])$ *as subgroups of* A. We can replicate the construction of a group isomorphism $\alpha : K(s,[f]) \cong K(s,[f'])$, given by a conjugation. Hasegawa notes that α is not canonical. In particular, α does not commute with the embeddings into A, hence these groups are not equal as subgroups.

We believe that translating Hasegawa's presentation of composites of analytic functors into the language of containers can either fix some of its shortcomings, or deduce precisely why action- or skeletal containers are not closed under substitution.

Embedding Type**-valued containers.** In order to understand precisely how containers embed into wild "functors" Type → Type, we would like to have an analogue of fully-faithfulness of $[\![\_]\!] : (0,0)\text{-Cont} \to \mathsf{Endo}(\mathsf{hSet})$. There is a short proof [DA23] of the latter employing the Yoneda lemma. The action on morphisms, sending container morphisms to natural transformations, decomposes as an equivalence

$$F \to G \simeq \left(\prod_{s:S} [\![G]\!](P_s)\right) \overset{\mathsf{yo}}{\simeq} \left(\prod_{s:S} (P_s \to \_) \Rightarrow [\![G]\!]\right) \simeq ([\![F]\!] \Rightarrow [\![G]\!])$$

for set-truncated containers $F \doteq (S \lhd P)$, $G \doteq (T \lhd Q)$. We are curious if this can be generalized to containers in arbitrary types, perhaps by parametricity assumption such as Lord's "easy parametricity" [Lor25].

Bibliography

[AAG05] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. "Containers: Constructing strictly positive types". In: *Theoretical Computer Science* 342.1 (2005), pp. 3–27. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2005.06.002 (cit. on pp. 32, 87, 121, 123).

[Abb+03] Michael Abbott, Thorsten Altenkirch, Neil Ghani, and Conor McBride. "Derivatives of Containers". In: *Proceedings of the 6th International Conference on Typed Lambda Calculi and Applications*. Lecture Notes in Computer Science 2701. Valencia, Spain, 2003, pp. 16–30. ISBN: 978-3-540-40332-6. DOI: 10.1007/3-540-44904-3_2 (cit. on p. 156).

[Abb+04] Michael Abbott, Thorsten Altenkirch, Neil Ghani, and Conor McBride. "Constructing Polymorphic Programs with Quotient Types". In: *Proc. of 7th Int. Conf. on Mathematics of Program Construction*. Lecture Notes in Computer Science 3125. 2004, pp. 2–15. ISBN: 978-3-54027764-4. DOI: 10.1007/978-3-540-27764-4_2 (cit. on pp. 3, 4, 69, 70, 72, 100, 101, 171).

[Abb+05] Michael Abbott, Thorsten Altenkirch, Conor McBride, and Neil Ghani. "∂ for Data: Differentiating Data Structures". In: *Fundamenta Informaticae* 65.1-2 (2005), pp. 1–28. DOI: 10.3233/FUN-2005-651-202 (cit. on pp. 5, 6, 117, 118, 137–140, 142, 143, 145, 148, 152, 156).

[Abb03] Michael Gordon Abbott. "Categories of Containers". PhD thesis. England, UK: University of Leicester, 2003. HDL: 2381/30102 (cit. on pp. 2, 5, 8, 31).

[ACH19] Jeremy Avigad, Mario Carneiro, and Simon Hudon. "Data Types as Quotients of Polynomial Functors". In: *16th International Conference on Interactive Theorem Proving (ITP'19)*. LIPIcs 141. 2019, 6:1–6:19. DOI: 10.4230/LIPICS.ITP.2019.6 (cit. on p. 3).

[ACS15] Benedikt Ahrens, Paolo Capriotti, and Régis Spadotti. "Non-Well-founded Trees in Homotopy Type Theory". In: *13th International Conference on Typed Lambda Calculi and Applications (TLCA'15)*. LIPIcs 38. 2015, pp. 17–30. DOI: 10.4230/LIPIcs.TLCA.2015.17 (cit. on pp. 4, 37, 51, 62, 66, 118, 128).

[Agd05] The Agda Developers. *Agda*. Version 2.8.0. 2005. URL: https://agda.readthedocs.io/ (cit. on p. 6).

[Agd25] The Agda Community. *Cubical Agda Library*. Version v0.9. 2025. URL: https://github.com/agda/cubical/ (cit. on p. 8).

[Ahr+21] Benedikt Ahrens, Dan Frumin, Marco Maggesi, Niccolò Veltri, and Niels van der Weide. "Bicategories in Univalent Foundations". In: *Mathematical Structures in Computer Science* 31.10 (2021), pp. 1232–1269. ISSN: 1469-8072. DOI: 10.1017/s0960129522000032 (cit. on pp. 28, 29, 31, 70).

[AL19] Benedikt Ahrens and Peter LeFanu Lumsdaine. "Displayed Categories". In: *Logical Methods in Computer Science* 15.20 (1 2019). ISSN: 1860-5974. DOI: 10.23638/lmcs-15(1:20)2019 (cit. on pp. 8, 26, 28).

[ALS10] Thorsten Altenkirch, Paul Blain Levy, and Sam Staton. "Higher-Order Containers". In: *6th Conf. on Computability in Europe (CiE'10)*. Lecture Notes in Computer Science 6158. 2010, pp. 11–20. DOI: 10.1007/978-3-642-13962-8_2 (cit. on pp. 2, 158).

[Alt+15] Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor McBride, and Peter Morris. "Indexed containers". In: *Journal of Functional Programming* 25 (2015). ISSN: 1469-7653. DOI: 10.1017/s095679681500009x (cit. on pp. 6, 48, 58, 118, 121, 123, 124, 128).

[AR20] Jiří Adámek and Jiří Rosický. "How nice are free completions of categories?" In: *Topology and its Applications* 273 (2020). ISSN: 0166-8641. DOI: 10.1016/j.topol.2019.106972 (cit. on pp. 86, 158).

[AT90] Jiří Adámek and Věra Trnková. *Automata and Algebras in Categories*. Mathematics and its Applications 37. Kluwer Acad. Publ., 1990. ISBN: 978-0-7923-0010-6 (cit. on pp. 47, 51).

[Bai+90] E. S. Bainbridge, P. J. Freyd, A. Scedrov, and P. J. Scott. "Functorial polymorphism". In: *Theoretical Computer Science* 70.1 (1990), pp. 35–64. ISSN: 0304-3975. DOI: 10.1016/0304-3975(90)90151-7 (cit. on pp. 2, 32).

[Bén85] Jean Bénabou. "Fibered categories and the foundations of naive category theory". In: *Journal of Symbolic Logic* 50.1 (1985), pp. 10–37. ISSN: 1943-5886. DOI: 10.2307/2273784 (cit. on p. 86).

[Bez+24] Marc Bezem, Ulrik Buchholtz, Pierre Cagne, Bjørn Ian Dundas, and Daniel R. Grayson. *Symmetry*. 2024. URL: https://github.com/UniMath/SymmetryBook/tree/85465273079c56b420a1cbd710ebf1d8b9c742db (visited on 2024-09-05) (cit. on pp. 104, 105, 112, 157).

[BLL97] François Bergeron, Gilbert Labelle, and Pierre Leroux. *Combinatorial Species and Tree-like Structures*. Cambridge University Press, 1997. ISBN: 9781107325913. DOI: 10.1017/cbo9781107325913 (cit. on p. 3).

[BMV22] Magnus Baunsgaard Kristensen, Rasmus Ejlers Mogelberg, and Andrea Vezzosi. "Greatest HITs. Higher inductive types in coinductive definitions via induction under clocks". In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'22)*. LICS '22. 2022, pp. 1–13. DOI: 10.1145/3531130.3533359 (cit. on p. 3).

[Bro06] Ronald Brown. *Topology and groupoids*. 3rd ed. BookSurge LLC, 2006. ISBN: 978-1-4196-2722-4. URL: https://groupoids.org.uk/topgpds.html (cit. on p. 3).

[Buc19] Ulrik Buchholtz. "Higher Structures in Homotopy Type Theory". In: *Reflections on the Foundations of Mathematics*. 2019, pp. 151–172. ISBN: 9783030156558. DOI: 10.1007/978-3-030-15655-8_7 (cit. on p. 119).

[CF23] Vikraman Choudhury and Marcelo Fiore. "Free Commutative Monoids in Homotopy Type Theory". In: *Proceedings of MFPS XXXVIII*. Vol. 1. 2023. DOI: 10.46298/entics.10492 (cit. on pp. 38, 39).

[CK17] Paolo Capriotti and Nicolai Kraus. "Univalent Higher Categories via Complete Semi-Segal Types". In: *Proceedings of the ACM on Programming Languages*. 51st ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2024). Vol. 2. 2017, pp. 1–29. DOI: 10.1145/3158132 (cit. on p. 22).

[CKV15] Paolo Capriotti, Nicolai Kraus, and Andrea Vezzosi. "Functions out of Higher Truncations". In: *24th EACSL Annual Conference on Computer Science Logic (CSL)*. LIPIcs 41. 2015, pp. 359–373. DOI: 10.4230/LIPIcs.CSL.2015.359 (cit. on p. 63).

[DA23] Stefania Damato and Thorsten Altenkirch. *Specifying QIITs using Containers*. Slides of a presentation at HoTT/UF'23. Vienna, 2023. URL: https://hott-uf.github.io/2023/slides/Damato.pdf (cit. on p. 160).

[DAL25] Stefania Damato, Thorsten Altenkirch, and Axel Ljungström. "Formalising Inductive and Coinductive Containers". In: *16th International Conference on Interactive Theorem Proving (ITP'25)*. LIPIcs 352. 2025, 17:1–17:20. DOI: 10.4230/LIPIcs.ITP.2025.17 (cit. on pp. 118, 124).

[dJon25] Tom de Jong. "Formalizing Equivalences Without Tears". In: *30th International Conference on Types for Proofs and Programs (TYPES'24)*. LIPIcs 336. 2025, 1:1–1:6. DOI: 10.4230/LIPIcs.TYPES.2024.1 (cit. on p. 13).

[dVL14] Edsko de Vries and Andres Löh. "True sums of products". In: *Proceedings of the 10th ACM SIGPLAN workshop on Generic programming*. ICFP'14. 2014, pp. 83–94. DOI: 10.1145/2633628.2633634 (cit. on p. 2).

[Dyb97] Peter Dybjer. "Representing inductively defined sets by wellorderings in Martin-Löf's type theory". In: *Theoretical Computer Science* 176.1–2 (1997), pp. 329–335. ISSN: 0304-3975. DOI: 10.1016/s0304-3975(96)00145-4 (cit. on p. 2).

[Ec10] Martín H. Escardó and contributors. *TypeTopology*. Agda development. 2010. URL: https://github.com/martinescardo/TypeTopology (visited on 2025-10-23) (cit. on pp. 8, 136).

[Fin+21] Eric Finster, Samuel Mimram, Maxime Lucas, and Thomas Seiller. "A Cartesian Bicategory of Polynomial Functors in Homotopy Type Theory". In: *Electronic Proceedings in Theoretical Computer Science* 351 (2021), pp. 67–83. ISSN: 2075-2180. DOI: 10.4204/eptcs.351.5 (cit. on pp. 18, 61, 62, 64).

[Fio+07] M. Fiore, N. Gambino, M. Hyland, and G. Winskel. "The cartesian closed bicategory of generalised species of structures". In: *Journal of the London Mathematical Society* 77.1 (2007), pp. 203–220. ISSN: 0024-6107. DOI: 10.1112/jlms/jdm096 (cit. on p. 3).

[Fru+18] Dan Frumin, Herman Geuvers, Léon Gondelman, and Niels van der Weide. "Finite Sets in Homotopy Type Theory". In: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs. CPP '18. 2018, pp. 201–214. DOI: 10.1145/3167085 (cit. on p. 18).

[Für+22] Basil Fürer, Andreas Lochbihler, Joshua Schneider, and Dmitriy Traytel. "Quotients of Bounded Natural Functors". In: *Logical Methods in Computer Science* 18 (1 2022). ISSN: 1860-5974. DOI: 10.46298/lmcs-18(1:23)2022 (cit. on p. 3).

[GK12] Nicola Gambino and Joachim Kock. "Polynomial functors and polynomial monads". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 154.1 (2012), pp. 153–192. ISSN: 1469-8064. DOI: 10.1017/s0305004112000394 (cit. on p. 3).

[Gyl11] Håkon Robbestad Gylterud. "Symmetric Containers". MA thesis. Department of Mathematics, Faculty of Mathematics and Natural Sciences, University of Oslo, 2011. HDL: 10852/10740 (cit. on pp. 3, 4, 32, 61, 69, 73, 114, 171).

[Has02] Ryu Hasegawa. "Two applications of analytic functors". In: *Theoretical Computer Science* 272.1–2 (2002), pp. 113–175. ISSN: 0304-3975. DOI: 10.1016/s0304-3975(00)00349-2 (cit. on pp. 41, 114, 158–160).

[Hed98] Michael Hedberg. "A coherence theorem for Martin-Löf's type theory". In: *Journal of Functional Programming* 8.4 (1998), pp. 413–436. ISSN: 1469-7653. DOI: 10.1017/s0956796898003153 (cit. on pp. 5, 11, 117).

[HK24] Pieter Hofstra and Martti Karvonen. "Inner automorphisms as 2-cells". In: *Theory and Applications of Categories* 42.2 (2024), pp. 19–40. URL: http://www.tac.mta.ca/tac/volumes/42/2/42-02abs.html (cit. on p. 90).

[Hue97] Gérard Huet. "The Zipper". In: *Journal of Functional Programming* 7.5 (1997), pp. 549–554. ISSN: 1469-7653. DOI: 10.1017/s0956796897002864 (cit. on p. 5).

[Joh21] Niles Johnson. *2-Dimensional Categories*. Description based on publisher supplied metadata and other sources. Oxford: Oxford University Press USA - OSO, 2021. ISBN: 978-0-19-264567-8 (cit. on p. 28).

[Joy81] André Joyal. "Une théorie combinatoire des séries formelles". In: *Advances in Mathematics* 42.1 (1981), pp. 1–82. ISSN: 0001-8708. DOI: 10.1016/0001-8708(81)90052-9 (cit. on p. 3).

[Joy86] André Joyal. "Foncteurs analytiques et espèces de structures". In: *Combinatoire énumérative*. Lecture Notes in Mathematics 1234. Universite du Quebec a Montreal. 1986, pp. 126–159. ISBN: 978-3-54047402-9. DOI: 10.1007/bfb0072514 (cit. on pp. 41, 71).

[Koc12] Joachim Kock. "Data Types with Symmetries and Polynomial Functors over Groupoids". In: *Electronic Notes in Theoretical Computer Science* 286 (2012), pp. 351–365. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2013.01.001 (cit. on pp. 3, 61, 157).

[Kra+17] Nicolai Kraus, Martín Escardó, Thierry Coquand, and Thorsten Altenkirch. “Notions of Anonymous Existence in Martin-Löf Type Theory”. In: *Logical Methods in Computer Science* 13.15 (1 2017). ISSN: 1860-5974. DOI: 10.23638/lmcs-13(1:15)2017 (cit. on pp. 117, 130).

[Kra15] Nicolai Kraus. “Truncation Levels in Homotopy Type Theory”. PhD thesis. University of Nottingham, Faculty of Science, School of Computer Science, 2015. Nottingham eTheses: 28986 (cit. on p. 19).

[Lab21] The 1Lab Development Team. *The 1Lab*. 2021. URL: https://1lab.dev (cit. on p. 8).

[Lab86] Gilbert Labelle. “On combinatorial differential equations”. In: *Journal of Mathematical Analysis and Applications* 113.2 (1986), pp. 344–381. ISSN: 0022-247X. DOI: 10.1016/0022-247x(86)90310-0 (cit. on p. 145).

[Lev11] Paul Blain Levy. “Similarity Quotients as Final Coalgebras”. In: *Foundations of Software Science and Computational Structures*. 2011, pp. 27–41. ISBN: 978-3-64219805-2. DOI: 10.1007/978-3-642-19805-2_3 (cit. on p. 40).

[LF14] Daniel R. Licata and Eric Finster. “Eilenberg–MacLane spaces in homotopy type theory”. In: *Proc. of the Joint Meeting of the 23rd EACSL Ann. Conf. on Computer Science Logic (CSL) and the 29th Ann. ACM/IEEE Symp. on Logic in Computer Science (LICS), CSL-LICS’14*. 2014, 66:1–66:9. DOI: 10.1145/2603088.2603153 (cit. on p. 74).

[Li15] Nuo Li. “Quotient types in type theory”. PhD thesis. University of Nottingham, Faculty of Science, School of Computer Science, 2015. Nottingham eTheses: 28941 (cit. on p. 42).

[Lor25] Jem Lord. “Easy Parametricity”. HoTT/UF’25. Genova, 2025. URL: https://hott-uf.github.io/2025/abstracts/HoTTUF_2025_paper_21.pdf (cit. on p. 160).

[Man88] Mark Mandelkern. “Constructively Complete Finite Sets”. In: *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 34.2 (1988), pp. 97–103. DOI: 10.1002/malq.19880340202 (cit. on p. 54).

[McB01] Conor McBride. “The Derivative of a Regular Type is its Type of One-Hole Contexts”. 2001. URL: http://strictlypositive.org/diff.pdf (visited on 2025-11-06) (cit. on pp. 5, 139).

[MM04] Conor McBride and James McKinna. “The view from the left”. In: *Journal of Functional Programming* 14.1 (2004), pp. 69–111. ISSN: 1469-7653. DOI: 10.1017/S0956796803004829 (cit. on p. 138).

[New+25] Max New, Steven Schaefer, Johnson He, Pranav Srinivasan, Nathan Varner, and Eric Bond. *Cubical Categorical Logic*. 2025. URL: https://github.com/um-catlab/cubical-categorical-logic (cit. on p. 8).

[Nyb22] Carl-Fredrik Nyberg-Brodda. "The Adian-Rabin Theorem. An English translation". In: (2022). DOI: 10.48550/ARXIV.2208.08560. arXiv: 2208.08560 [math.GR] (cit. on p. 158).

[Pic21] Stefano Piceghello. "Coherence for Monoidal and Symmetric Monoidal Groupoids in Homotopy Type Theory". PhD thesis. The University of Bergen, 2021. HDL: 11250/2830640 (cit. on p. 65).

[PP99] D. Pavlović and V. Pratt. "On coalgebra of real numbers". In: *Electronic Notes in Theoretical Computer Science* 19 (1999), pp. 103–117. ISSN: 1571-0661. DOI: 10.1016/s1571-0661(05)80272-5 (cit. on p. 54).

[Rij+21] Egbert Rijke, Elisabeth Stenholm, Jonathan Prieto-Cubides, Fredrik Bakke, et al. *The agda-unimath library*. 2021. URL: https://github.com/UniMath/agda-unimath/ (cit. on p. 8).

[Rij17] Egbert Rijke. "The join construction". In: (2017). arXiv: 1701.07538 [math.CT] (cit. on p. 19).

[RSS20] Egbert Rijke, Michael Shulman, and Bas Spitters. "Modalities in homotopy type theory". In: *Logical Methods in Computer Science* 16 (1 2020). ISSN: 1860-5974. DOI: 10.23638/lmcs-16(1:2)2020 (cit. on p. 14).

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory. Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013. URL: https://homotopytypetheory.org/book (cit. on pp. 8, 12, 14, 20, 21, 25, 26, 74, 76, 80, 90, 95, 103, 119, 120).

[vdWG19] Niels van der Weide and Herman Geuvers. "The Construction of Set-Truncated Higher Inductive Types". In: *Electronic Notes in Theoretical Computer Science* 347 (2019), pp. 261–280. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2019.09.014 (cit. on p. 19).

[Vel17] Niccolò Veltri. "A Type-Theoretical Study of Nontermination". Tallinn University of Technology, 2017. ISBN: 978-9949-83100-5. URL: https://digi.lib.ttu.ee/i/?7631 (cit. on p. 60).

[Vel21] Niccolò Veltri. "Type-Theoretic Constructions of the Final Coalgebra of the Finite Powerset Functor". In: *6th International Conference on Formal Structures for Computation and Deduction (FSCD'21)*. LIPIcs 195. 2021, 22:1–22:18. DOI: 10.4230/LIPIcs.FSCD.2021.22 (cit. on pp. 48, 54, 61).

[VMA21] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. "Cubical Agda: A dependently typed programming language with univalence and higher inductive types". In: *Journal of Functional Programming* 31 (2021). ISSN: 1469-7653. DOI: 10.1017/s0956796821000034 (cit. on p. 6).

[Voe15] Vladimir Voevodsky. "An experimental library of formalized Mathematics based on the univalent foundations". In: *Mathematical Structures in Computer Science* 25.5 (2015), pp. 1278–1294. DOI: 10.1017/s0960129514000577 (cit. on pp. 1, 8, 12).

[VvdW21] Niccolò Veltri and Niels van der Weide. "Constructing Higher Inductive Types as Groupoid Quotients". In: *Logical Methods in Computer Science* 17.8 (2 2021). ISSN: 1860-5974. DOI: 10.23638/lmcs-17(2:8)2021 (cit. on p. 19).

[Wel76] Charles Wells. "Some Applications of the Wreath Product Construction". In: *The American Mathematical Monthly* 83.5 (1976), pp. 317–338. ISSN: 1930-0972. DOI: 10.1080/00029890.1976.11994114 (cit. on p. 114).

[Wor05] James Worrell. "On the final sequence of a finitary set functor". In: *Theoretical Computer Science* 338.1–3 (2005), pp. 184–199. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2004.12.009 (cit. on p. 48).

[Yor14] Brent Yorgey. "Combinatorial Species and Labelled Structures". PhD thesis. University of Pennsylvania, 2014. URL: http://ozark.hendrix.edu/~yorgey/pub/thesis.pdf (cit. on p. 145).

[Yua14] Qiaochu Yuan. *Describing the Wreath product categorically*. 2014. Mathematics Stack Exchange: https://math.stackexchange.com/q/867292 (cit. on p. 90).

Acknowledgements

First and foremost, thank you Niccolò for taking me on as your student. Under your guidance I have gained a lot of confidence in my own abilities, and have grown as an independent researcher. You have always entertained my vague ideas, and pushed me to see them through.

I am grateful for the numerous discussions I have had with my colleagues at the High-assurance Software Laboratory as well as those of our sister group of Compositional Systems and Methods. I would like to thank Tarmo for being a dependable group leader, and my PhD siblings for a lot of shared fun. I am indebted to the reviewers of this thesis whose feedback has improved it in many places, as well as Nathanael for his attention to typographical detail.

I thank my parents, Anke & Thomas, for setting me on this path. You have always encouraged my curiosity and supported me in any way possible to reach my goals. Thank you Ele for being a kind partner and an amazing friend. You have shown a lot of patience and have taken care of me when I was busy taking care of this document. Last but not least, thank you Kohupiim, my furry friend, for enforcing writing breaks by sitting on my keyboard.

My research and participation in workshops and conferences has been supported by the Estonian Research Council grants PSG749 and PSG659, and the EuroProofNet EU COST Action CA20111.

Abstract

We study datatypes with symmetries in homotopy type theory by representing them as containers.

We show that definitions of such containers based on set-truncated types (h-sets) are insufficient to constructively model non-wellfounded datatypes: Typically, non-wellfounded datatypes are interpreted as final coalgebras of endofunctors of sets. Classically, the type of non-wellfounded trees with finite and unordered branching is the final coalgebra of the finite multiset functor. We prove that this coalgebra is final if and only if the constructively non-derivable lesser limited principle of omniscience holds.

Gylterud's symmetric containers are based on groupoid-truncated types, and are known to induce final coalgebras in a suitable 2-categorical way. Finite multisets are represented both by a quotient container (in the sense of Abbott et al.) as well as a symmetric container. We make this translation from quotient- to symmetric containers precise. To this end we introduce action containers, a variation of the quotient containers satisfying additional coherence conditions. We construct a locally fully-faithful functor from the 2-category of action containers into that of symmetric containers. This 2-functor restricts to an equivalence between a 1-category of action containers and symmetric containers whose groupoids of shapes carry additional structure. From this 1-categorical semantics we derive a partial substitution operation for action containers, previously left undefined for quotient containers.

Lastly, we define a derivative operation for containers whose shapes and positions are arbitrary, untruncated types. It satisfies a universal property with respect to all containers and cartesian morphisms. We derive a chain rule and prove that is an embedding of containers. For groupoid-truncated containers, the chain rule may fail to be an equivalence. For set-truncated containers, it is an equivalence if and only if arbitrary h-sets have decidable equality. To prove these results, we establish properties of the isolated points of arbitrary types, and how isolatedness distributes over various type formers. In particular, we characterize the isolated points of dependent sums in terms of non-constructive axioms.

Kokkuvõte

Käesolevas töös uurime sümmeetriatega andmetüüpe homotoopilises tüübiteoorias, esitades neid konteineritena.

Näitame, et hulkadeks trunkeeritud tüüpidel (h-hulkadel) põhinevate konteineritega pole mittefundeeritud andmetüüpide konstruktiivne modelleerimine võimalik. Tavaliselt interpreteeritakse mittefundeeritud andmetüüpe hulkade endofunktorite terminaalsete koalgebratena. Klassikaliselt on mittefundeeritud lõplikult hargnevad järjestamata puud lõplike multihulkade funktori terminaalne koalgebra. Me näitame, et see koalgebra on terminaalne parajasti konstruktiivselt tuletamatu väikese piiratud omnistsientsi printsiibi eeldusel.

Gylterudi sümmeetrilised konteinerid põhinevad rühmoididdeks trunkeeritud tüüpidel. On teada, neil on sobivas 2-kategooriateoreetilises tähenduses olemas terminaalsed koalgebrad. Lõplikud multihulgad on esitatavad nii faktorkonteineri (Abbotti jt. mõttes) kui ka sümmeetrilise konteinerina. Me selgitame faktorkonteinerite ja sümmeetriliste konteinerite vahekorra. Selleks toome sisse toimekonteinerid kui täiendavaid koherentsitingimusi rahuldavad faktorkonteinerid. Me konstrueerime lokaalselt täpse funktori toimekonteinerite 2-kategooriast sümmeetriliste konteinerite omasse. See funktor aheneb ekvivalentsiks toimekonteinerite 1-kategooria ja niisuguste sümmeetriliste konteinerite 1-kategooria vahel, mille kujude rühmoididel on teatud lisastruktuur. Sellest 1-kategoorsest semantikast tuletame toimekonteinerite jaoks osalise substitutsioonioperaatori, mida varem pole käsitletud.

Lõpuks üldistame tuletise operatsiooni konteinerite jaoks, mille kujud ja positsioonid on trunkeerimata tüübid. Sellel operatsioonil on universaalomadus kõigi konteinerite ja Descartes'i konteinerimorfismide suhtes. Me tuletame ahelreegli ja näitame, et ta on sisestus. Rühmoididdeks trunkeeritud konteinerite puhul jaoks võib ta olla mitte-ekvivalents. Hulkadeks trunkeeritud konteinerite korral on ta ekvivalents parajasti võrduse lahenduvuse eeldusel kõigi h-hulkade jaoks. Nende tulemuste tõestamiseks teeme kindlaks suvaliste tüüpide isoleeritud punktide mõned omadused ja kuidas isoleeritus jaotub üle erinevate tüübimoodustajate. Muu hulgas karakteriseerime me sõltuvate summade isoleeritud punktid mittekonstruktiivsete aksioomide kaudu.

Curriculum Vitae

Name	Philipp Joram
Birthdate and -place	26 August 1996, Plauen, Germany
Nationality	German
Languages	English *(fluent)*, German *(native)*

Education

PhD in *Information and Communication Technology* 2021–*present*
- Institution: Tallinn University of Technology
- Thesis: *Datatypes with Symmetries in Homotopy Type Theory*
- Supervision: Niccolò Veltri

MSc in *Mathematics* 2018–2021
- Institution: Dresden Technical University
- Thesis: *A Geometric Approach to Cyclic Duality*
- Supervision: Ulrich Krämer

BSc in *Mathematics* 2015–2018
- Institution: Dresden Technical University
- Thesis: *Operational Semantics by Example: The Programming Language Loop*
- Supervision: Ulrich Krämer

Employment

Early Stage Researcher 2021–*present*
- Department of Software Science, Tallinn University of Technology

Research Assistant 2017–2021
- Project *HEAC*, ZIH Dresden, Technical University Dresden

Teaching

Teaching Assistant 2023
- *Software Quality & Standards (IDY0204)*, Tallinn University of Technology

Teaching Assistant 2022
- *Functional Programming (ITI0212)*, Tallinn University of Technology

Teaching Assistant 2020
- *Representation Theory of Finite Groups*, AIMS Ghana, Accra

Publications

1. Philipp Joram and Niccolò Veltri. "Data Types with Symmetries via Action Containers". In: *30th International Conference on Types for Proofs and Programs (TYPES'24)*. LIPIcs 336. 2025, 6:1–6:21. DOI: 10.4230/LIPIcs.TYPES.2024.6
2. Tavo Annus and Philipp Joram. "Term Search in Rust". In: *Proceedings of the 9th ACM SIGPLAN International Workshop on Type-Driven Development*. TyDe '24. 2024, pp. 62–73. DOI: 10.1145/3678000.3678210
3. John Boiquaye, Philipp Joram, and Ulrich Krähmer. "Cyclic duality for slice and orbit 2-categories". In: *Higher Structures* 8.2 (2024), pp. 136–162. DOI: 10.21136/hs.2024.09
4. Philipp Joram and Niccolò Veltri. "Constructive Final Semantics of Finite Bags". In: *14th International Conference on Interactive Theorem Proving (ITP'23)*. LIPIcs 268. 2023, 20:1–20:19. DOI: 10.4230/LIPIcs.ITP.2023.20
5. Thomas Ilsche, Robert Schöne, Philipp Joram, Mario Bielert, and Andreas Gocht. "System Monitoring with lo2s: Power and Runtime Impact of C-State Transitions". In: *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2018, pp. 712–715. DOI: 10.1109/ipdpsw.2018.00114

Elulookirjeldus

Nimi	Philipp Joram
Sünniaeg ja -koht	26. august 1996, Plauen, Saksamaa
Kodakondsus	saksa
Keelteoskus	inglise *(kõrgtase)*, saksa *(emakeel)*

Haridustee

Doktoriõpingud *info- ja kommunikatsioonitehnoloogias* 2021–*praeguseni*
- asutus: Tallinna Tehnikaülikool
- lõputöö: *Sümmeetriatega andmetüübid homotoopilises tüübiteoorias*
- juhendajad: Niccolò Veltri

Magistriõpingud *matemaatikas* 2018–2021
- asutus: Dresdeni Tehnikaülikool
- lõputöö: *Geomeetriline lähenemine tsüklilisele duaalsusele*
- juhendajad: Ulrich Krämer

Bakalaureusõpingud *matemaatikas* 2015–2018
- asutus: Dresdeni Tehnikaülikool
- lõputöö: *Operatsiooniline semantika programmeerimiskeele Loop näitel*
- juhendajad: Ulrich Krämer

Teenistuskäik

Doktorant-nooremteadur 2021–*praeguseni*
- Tarkvarateaduse instituut, Tallinna Tehnikaülikool

Teadusassistent 2017–2021
- Projekt *HEAC*, ZIH Dresden, Dresdeni Tehnikaülikool

Õppetöö

Õppeassistent 2023
- *Tarkvara kvaliteet ja standardid (IDY0204)*, Tallinna Tehnikaülikool

Õppeassistent 2022
- *Funktsionaalprogrammeerimine (ITI0212)*, Tallinna Tehnikaülikool

Õppeassistent 2020
- *Lõplike rühmade esitusteooria*, AIMS Ghana, Accra

Publikatsioonid

vt ingliskeelset CV-d

ISSN 2585-6901 (PDF)
ISBN 978-9916-80-515-2 (PDF)