Coherence for Monoidal and Symmetric Monoidal Groupoids in Homotopy Type Theory

Stefano Piceghello

Thesis for the degree of Philosophiae Doctor (PhD)
University of Bergen, Norway
2021

UNIVERSITY OF BERGEN

Coherence for Monoidal and Symmetric Monoidal Groupoids in Homotopy Type Theory

Stefano Piceghello

Thesis for the degree of Philosophiae Doctor (PhD)
at the University of Bergen

Date of defense: 03.12.2021

Year: 2021

Title: Coherence for Monoidal and Symmetric Monoidal Groupoids in Homotopy Type Theory

Name: Stefano Piceghello

Print: Skipnes Kommunikasjon / University of Bergen

Perché, qualunque cosa scriva in questi mesi di ozio febbrile, sarà sempre soltanto una «curiosità» per il futuro, cioè silenzio. Cadono in questi mesi molti valori del passato e si distruggono abitudini interiori, che – straordinaria fortuna – nulla per ora sostituisce. [...] Passare ore a rosicchiarmi le unghie, a disperare degli uomini, a disprezzare luce e natura, a temere per paure infantili e pure atroci, è un ritorno ai miei vent'anni. Quale mondo giaccia di là di questo mare non so, ma ogni mare ha l'altra riva, e arriverò.

C. Pavese, *Il mestiere di vivere (diario 1935–1950)*, 1952

Abstract

Homotopy Type Theory (HoTT) is a variant of Martin-Löf Type Theory (MLTT) developed in such a way that types can be interpreted as ∞-groupoids, where the iterated construction of identity types represents the different layers of higher path space objects. HoTT can be used as a foundation of mathematics, and the proofs produced in its language can be verified with the aid of specific proof assistant software. In this thesis, we provide a formulation and a formalization of coherence theorems for monoidal and symmetric monoidal groupoids in HoTT.

In order to design 1-types $\mathsf{FMG}(X)$ and $\mathsf{FSMG}(X)$ representing the free monoidal and the free symmetric monoidal groupoid on a 0-type X of generators, we use higher inductive types (HITs), which apply the functionality of inductive definitions to the higher groupoid structure of types given by the identity types. Coherence for monoidal groupoids is established by showing a monoidal equivalence between $\mathsf{FMG}(X)$ and the 0-type $\mathsf{list}(X)$ of lists over X. For symmetric monoidal groupoids, we prove a symmetric monoidal equivalence between $\mathsf{FSMG}(X)$ and a simpler HIT $\mathsf{slist}(X)$ based on lists, whose paths and 2-paths make for an auxiliary symmetric structure on top of the monoidal structure already present on $\mathsf{list}(X)$.

Part of the thesis is devoted to the proof that the subuniverse $\mathcal{B}\mathsf{S}_\bullet$ of finite types is equivalent to the type $\mathsf{slist}(\mathbf{1})$, where $\mathbf{1}$ is the unit type, and hence that the former is a free symmetric monoidal groupoid. As an intermediate step, we show a symmetric monoidal equivalence between $\mathsf{slist}(\mathbf{1})$ and an indexed HIT $\mathsf{del}_\bullet$ of deloopings of symmetric groups. The proof of a symmetric monoidal equivalence between $\mathsf{del}_\bullet$ and $\mathcal{B}\mathsf{S}_\bullet$ rests on a few, unformalized statements. Assuming this equivalence, we are able to prove that, in a free symmetric monoidal groupoid, all diagrams involving symmetric monoidal expressions without repetitions commute.

This work is accompanied by a computer verification in the proof assistant Coq, which covers most of the results we present in this thesis.

Acknowledgements

First and foremost, I would like to thank my supervisors Bjørn Ian Dundas and Marc Bezem, who taught me, throughout the years, about mathematics, type theory, and life. Their support and the friendly and cohesive environment they managed to create and cultivate were decisive in all stages of the production of this thesis, and I hope to have the chance to work with them again in the future.

Countless times did I share ideas with the people belonging to or gravitating around the CAU project at the University of Bergen. I'm grateful in particular to Håkon R. Gylterud, Kristian Alfsvåg, Pierre Cagne, Jonathan Prieto-Cubides, Robin Adams and Andrew Polonsky for always listening and for their contribution on numerous aspects of the theory presented in this thesis.

I am indebted to Peter Dybjer, Floris van Doorn, Favonia and Ulrik Buchholtz for sharing their invaluable insight on coherence theorems, expertise on formalization and refined knowledge about type theory.

I'm happy to have been part of the Topology Group at the University of Bergen, and I'm grateful for the assistance received by the administrative staff at the Department of Mathematics and the Department of Informatics. I would also like to thank the administrative staff at the Centre for Advanced Study in Oslo for making me feel welcome during my time in the city.

Many friends connected to the Department of Mathematics made my years as a Ph.D. student in Bergen a truly enjoyable experience. I'd like to thank Jakub, Evgueni and Parisa, who turned a small office into a great working space, and Tommy, Andrea, Francesca, Eugenia, Erlend, Erlend, Valentin, Nazanin, Mirjam, Anastasia and Victor for the many nice moments spent together. Despite the physical distance and the circumstances which scattered us all around Europe, my friends Stefano, Giulia, Igina, Allegra, Veronica, Sofia, Valentina, Andrea, Simone and Silvia were with me in this journey from our time as students in Padova – or earlier – until now, and they deserve to be thanked for sticking around for so long.

I thank my family in Italy and my family in the Netherlands for their continued support, and Wietse, for making the world around me

a wonderful place,

every day.

*The author acknowledges the support of the Centre for Advanced Study (CAS) in Oslo, Norway, which funded and hosted the research project "*Homotopy Type Theory and Univalent Foundations*" during the 2018/19 academic year.*

in loving memory of
Clara

11.08.1925 — 30.01.2019

Contents

List of Figures and Tables

Chapter 1

Introduction

In the words of Leinster [Lei04], a coherence theorem in category theory is

> «*roughly, a description of a structure that makes it more manageable*».

The purpose of this thesis is to provide a formulation and a formalization of coherence theorems for monoidal and symmetric monoidal groupoids in Homotopy Type Theory.

1.1 Homotopy Type Theory

Homotopy Type Theory [Uni13] is the name given to a variant of Martin-Löf Type Theory (MLTT) invented and developed in recent years. A type theory is a formal system in logic, i.e., a system of axioms and rules that can be used to derive theorems. The theoretical apparatus of MLTT may be used as a foundational backbone for mathematics: this means that it provides a language and an abstract environment by which certain mathematical objects, statements and proofs can be encoded, communicated and, ultimately, understood by humans.

Constructivism and Proof Relevance

The type theory presented by Martin-Löf dates back to the 1970s [M-L75]. It is also known as "intuitionistic type theory", implying its connection to *intuitionism*, a philosophical approach to mathematics which constitutes one of the many incarnations of constructivism, and whose principles were laid down in the first half of the 20th century in the work of Brouwer [in e.g. Bro07; Bro08; see also the translation in vAS15; and Tro11, for an account of the history of constructivism]. The paradigm in which MLTT operates then sees mathematics as an activity consisting of building mathematical *constructions*. From this viewpoint, all that can be produced mathematically falls into the idea of a construction – whether it be the list of instructions required to draw a geometric figure; the definition of an operation between sets; a

sorting algorithm; an exact sequence of groups; a simplicial complex; a morphism in a category defined by means of a universal property, or any other object (Fig. 1.1).

Significantly, mathematical *proofs* are also considered constructions. A constructive approach rejects an aprioristic endorsement of those proof techniques – dubbed "classical", or nonconstructive – which do not exhibit an explicit construction, such as proofs by contradiction or, in general, proofs invoking the law of excluded middle. Admitting the use of these techniques is akin to the inclusion of additional hypotheses to the premises of a theorem: they must be declared, and their employment makes for a weaker result. This idea is closely linked to the concept of *proof relevance*: a proven theorem is not just considered "true", but its validity is established by the specific proof that substantiates the claim it contains. In other words, it may be true in essentially different ways; a statement claiming the existence of a natural number divisible by 2 has, for instance, as many proofs as there are even natural numbers. Other mathematical constructions – such as theorems that rest on previously established results – may then depend on the exact proof of those statements.

A Language for Constructions

MLTT is a language for mathematical constructions. The parts of speech of this language – and of any type theory – are entities labelled *types* and *terms*. Types encode classes of mathematical constructions, such as sets, spaces, or mathematical statements. In turn, terms encode instances of such constructions (elements of a set, points in a space, proofs of a statement) and are invariably "typed": they never occur autonomously, but always as terms *of* some type. Both types and terms manifest themselves in the form of expressions (words) which constitute the lexical units in the language.

Typed expressions are commonly found in a wide range of habitats. They are used in programming (for example, if there is need for an input to be considered a string rather than a number), in mathematics (when we fail to interpret the expression "$g \circ f$" as the composition of the functions f and g, if the target of f does not match the source of g) and even in everyday life (giving us the ability to parse an expression such as `(+47) 555 80 000` and recognize it as a telephone number). In the formal setting given by MLTT we might have, for instance, a type T whose terms are specific instances of triangles, or the type $\mathbb{N}$ of natural numbers, but also the cartesian product type $\mathbb{N} \times \mathbb{N}$ of pairs of such numbers, and the function type $(\mathbb{N} \times \mathbb{N}) \to \mathbb{N}$ of binary operations on natural numbers.

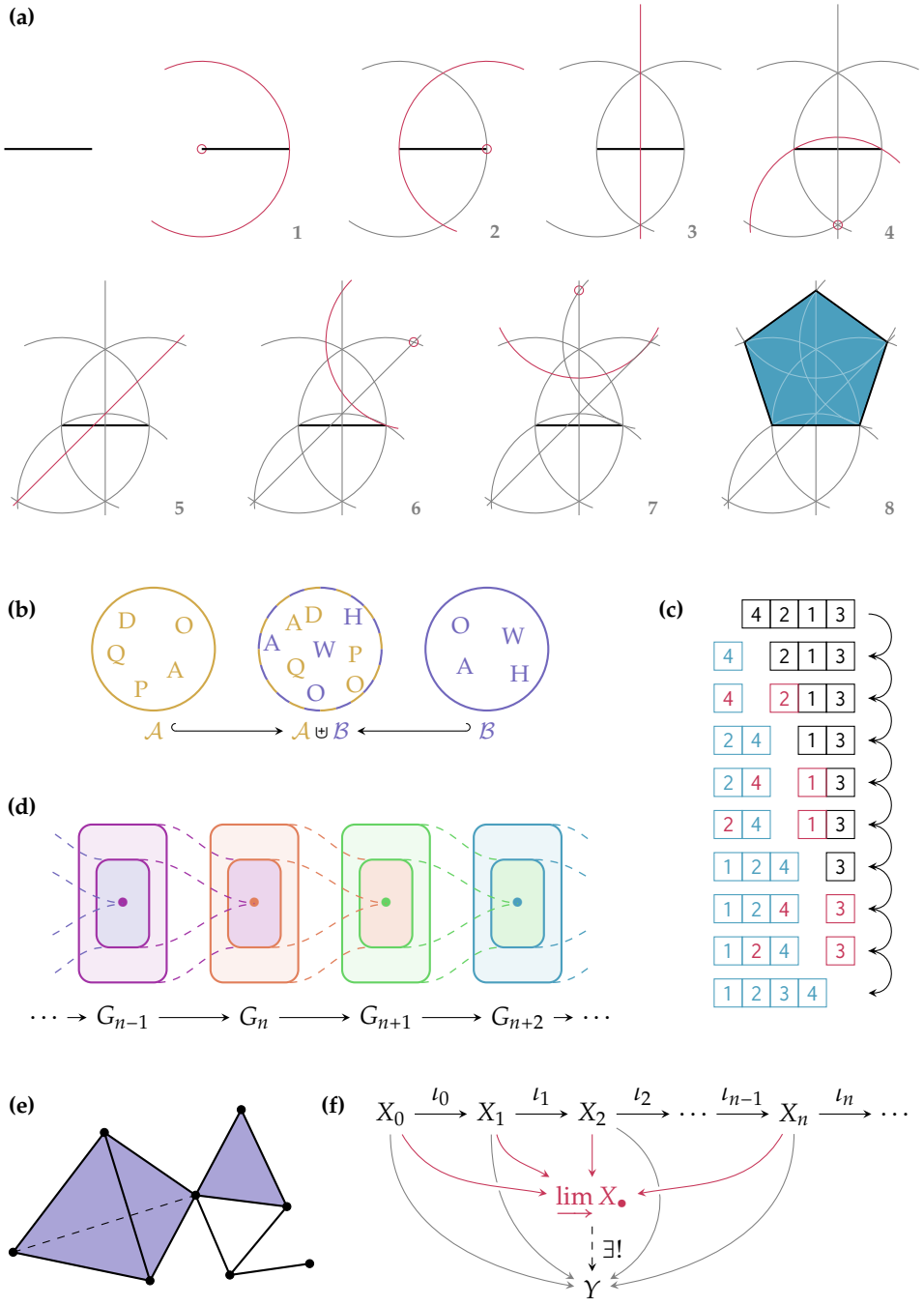


Figure 1.1: An array of mathematical constructions: **(a)** the classical construction of a pentagon using straightedge and compass; **(b)** the disjoint union of two sets; **(c)** iteration of the sorting algorithm *insertion sort* on a list; **(d)** visualization of an exact sequence of groups; **(e)** a simplicial complex; **(f)** direct limit of a direct system and its universal property.

Similarly, we could have a type P representing the statement «the sum of the angles of any triangle is π»; this may have, as terms, specific proofs of such a claim. The relationship between types/terms and statements/proofs adheres to what is known as the *Curry-Howard interpretation* [CFC58]: if the types P and Q represent the propositions P and Q, then the logical conjunction "P and Q" corresponds to the cartesian product type $P \times Q$, and the implication "if P then Q" matches the function type $P \to Q$. Other types can be formed, further expanding this analogy; these are interpreted to the logical disjunction of propositions, tautologies, contradictions, negation, and so forth. A counterpart in MLTT is also found for statements containing universal or existential quantifiers, because the language allows the presence of *families* of types, which take the role of the predicates of such propositions. These are called *dependent types*, and thus MLTT is a *dependent type theory*.

In the language given by the type theory, types and terms are used to form sentences. These are called *judgments* and make a variety of assertions, indicating for example that some expression is a type, or a term of some specified type. Rules are provided on how to derive such judgments, and in particular on how to build types and their terms. Moreover, MLTT possesses a computational aspect concerning the syntax of its terms, by which these can be *rewritten* into different (simpler) forms. For instance, rewriting can be used to evaluate numerical expressions in $\mathbb{N}$, as in

$$2 \times (3+4) \rightsquigarrow 2 \times 7 \rightsquigarrow 14 \qquad \text{or} \qquad 9+5 \rightsquigarrow 14.$$

A judgment of the theory will then pronounce the terms $2 \times (3+4)$, $9+5$ and 14 *judgmentally equal* (or computationally equal) terms of the same type $\mathbb{N}$.

The kind of judgmental equality illustrated above reflects the computational behaviour of terms, but does not express a construction. Stating that $2 \times (3+4)$ and $9+5$ compute to the same term is indeed a *judgment* of the theory, and not, in itself, a proof that they are "equal": this, we recall, would ask instead for the construction of a *term* in some type. Since expressing mathematical equality is a desirable feature for a language for mathematical constructions, a class of types that can be formed is that of *identity types* between terms (necessarily of the same type). Identity types are the theory's way of internalizing the idea of "sameness"; for every two terms x and y in the same type, there is a type $x = y$ whose terms (possibly, none) are proofs of identity between them. For example, for any two pairs (a,b) and (c,d) of natural numbers, a proof of their identity is a term in $(a,b) = (c,d)$, which can be built using a proof of $a = c$ and a proof of $b = d$. The fact that judgmental equality and proofs of identity are kept as separate notions makes MLTT an *intensional type*

theory, as opposed to other type theories where these two levels of equality are merged, which are called *extensional*.

Features of HoTT

The nature of identity types plays a central role in the way Homotopy Type Theory (HoTT) expands MLTT. Simple versions of MLTT maintain the position that any two terms of an identity type are, themselves, identified: for terms x and y of a type X, and for terms p and q of the type $x = y$, one postulates a term of the type $p = q$. This feature is known as *uniqueness of identity proofs* (UIP), and it is usually implemented as an assumption – often left implicit – called *axiom K* [Str93].

HoTT does not assume UIP; instead, it takes the stance that a sensible notion of equality should reflect the intuition behind *homotopy equivalences*, and it accepts the idea that, from different proofs of identity, fundamentally different constructions might arise. For instance, two finite sets could be identified whenever they have the same number of elements, and the proof of identity should carry the information regarding how this identification takes place, i.e., which bijection is used. The choice of different bijections might relate to constructions that ought not to be identified, such as the two covering spaces of the circle in Fig. 1.2. The marriage – itself an equivalence, in fact – between the notion of equality and that of equivalence is established by an axiom introduced by Voevodsky, called the *univalence axiom* [Voe14; KL18; Uni13]. The pursuit of a (computer-verified) foundation of mathematics by

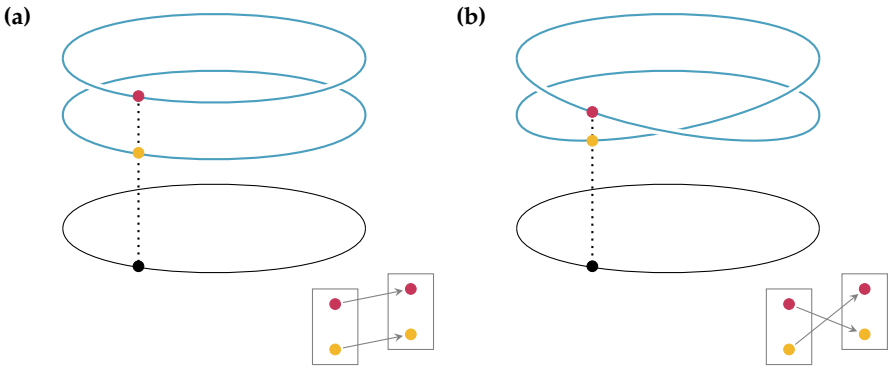


Figure 1.2: Two covering spaces of the circle. Both of them have two sheets, and hence they both are families of sets with two elements, parametrized by the circle. However, the covering space in **(a)** consists of two copies of the circle, while the one in **(b)** consists of one copy only. This reflects the fact that there are (two) different ways to identify a two-element set with itself.

means of a type theory obeying this axiom is called "*univalent foundations*", a designation that nowadays is often used liberally to refer to HoTT itself.

Not constrained by UIP, identity types in HoTT possess a less rigid, more sophisticated structure, which digs deeper into the mathematical concept of equality and calls *higher* identity types into existence: the types of identifications between identifications need not be trivial, and the ensuing tower of identity types can grow unbounded (Fig. 1.3). This allows a homotopical interpretation of types – a feature envisioned early in the history of HoTT as a field of research [HS98; AW09; Voe06], which gives the name to this type theory and which we find, in some aspects, well suited to the formalization of the results contained in this thesis. In this interpretation, terms p and q of a type $x = y$ are seen as *paths* between x and y; accordingly, terms of type $p = q$ are *homotopies* between p and q, while types further up in the tower of identity types may contain higher homotopies.

Collectively, terms in identity types satisfy ∞-groupoid axioms and laws. There is always a "unit" term in every type $x = x$ (the trivial path), along with an inverse operation $(-)^{-1} : (x = y) \to (y = x)$ (producing inverse paths), and an operation $\cdot : (x = y) \times (y = z) \to (x = z)$ (concatenation of paths, showing the transitivity property of equality). Associativity and inverse laws inhabit the realm of higher homotopies and are represented by terms in identity types built upon identity types, such as $(p \cdot q) \cdot r = p \cdot (q \cdot r)$. Coherence of these laws, in a sense explained in the next section, is also contemplated, making types akin to higher groupoids. For this reason, HoTT has been also defined a "synthetic theory of ∞-groupoids" [Shu17b], in which such objects, along with the notions of paths and homotopies, are intended to be primitive and not bound ("analytically") to any specific model or presentation – for example, by interpreting types as particular topological spaces, or paths as continuous functions out of the topological unit interval $[0,1]$. The claim is that

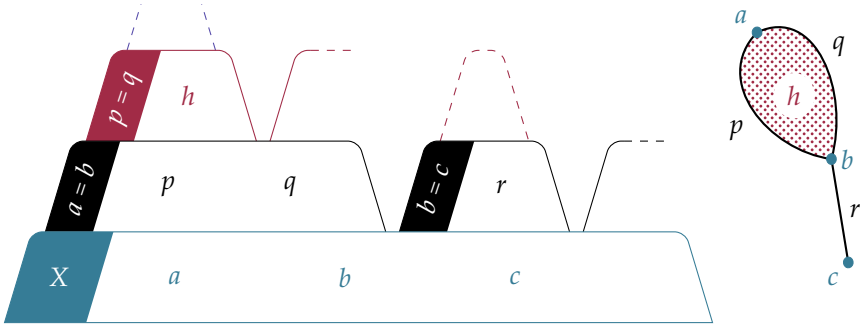


Figure 1.3: Towers of identity types: a visual representation.

∞-groupoids could replace sets in the foundation of mathematics.

As one would expect, constructions and statements that depend on terms of a certain type respect identities in that type. For instance, the family of types encoding the open statement "the natural number x is even" is dependent on a free variable x in the type $\mathbb{N}$. If n and m are terms in $\mathbb{N}$, a path in $n = m$ will make it possible for a proof that n is even to be translated to a proof that m is too: thus, the notion of identity in HoTT conforms to the principle of *indiscernibility of identicals*. This simple fact, combined with the univalence axiom (which merges the notions of equality and of equivalence of types), has the profound consequence that all constructions in HoTT are *homotopy invariant*: an equivalence of two types induces an identity between them, and hence all that can be stated about one of them holds for the other one too. For example, the one-term type ("the point") and the interval type (with two terms and a path between them; Fig. 1.4) are equivalent types (indeed, as spaces, the former is a deformation retract of the latter), so they are effectively indistinguishable within the theory.

An effective way of exploiting the nontrivial structure of identity types is by means of the powerful tool of *higher inductive types* (HITs), which allow the definition of types (freely) generated by terms, paths and higher homotopies; the interval type, mentioned above, is an example of a type that can be expressed as a HIT. Definitions of this kind are commonplace in mathematics. For instance, the suspension ΣX of a space X is obtained as the homotopy pushout of the cone $\bullet \leftarrow X \rightarrow \bullet$, i.e., it consists of two points joined by a family of paths, which in some precise sense takes into account the shape of X (see Fig. 1.4). The title "inductive" for such types

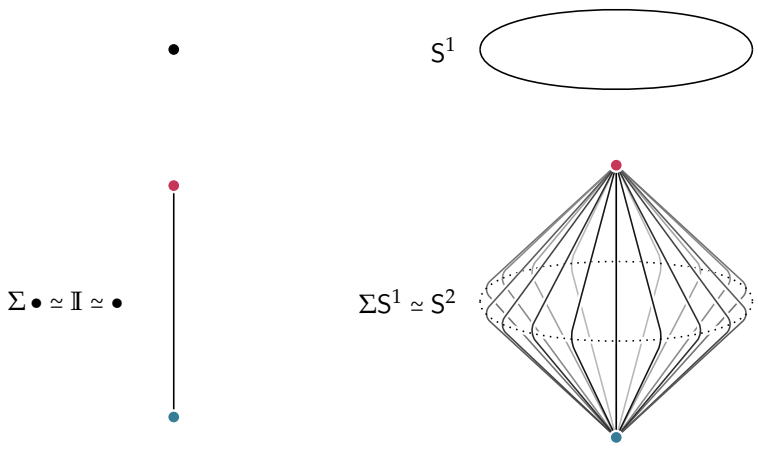


Figure 1.4: The suspension of the point and of the circle.

is to signify that their definition contains the universal property that the construction should satisfy (in our example, what it takes to build a map out of ΣX). The history of how HITs came to be is detailed in [Uni13]. The semantic interpretation of HITs is subject of current research [see e.g. LS19]; in this thesis we will set aside all issues regarding the semantics of HITs and focus on results that can be produced *internally* in a type theory that supports the HITs we need.

Several expansions or variations to the core of HoTT are object of ongoing research. Some of these are: Real-Cohesive HoTT [Shu17a], which is suited for differential geometry; Cubical Type Theory [Coh+18], which possesses advantageous computational features and investigates in particular the computational content of the univalence axiom [BCH14]; and a Two-Level Type Theory encompassing HoTT [Ann+19]. The latter is connected to the definition of semi-simplicial types and the problem of handling, internally to the theory, constructions entailing an infinite amount of coherence data, such as ∞-groupoids or A_∞-spaces – which we will encounter further on in this introduction.

Computer-Verified Proofs

Type theory is well suited for computer formalization [see e.g. NG14, Chapter 16], which can be performed by means of *proof assistants*. This kind of software allows the user to verify the correctness of the proofs one aims to construct. Usually, this is done interactively: the user specifies a goal (i.e., finding a term in a type), which can be then simplified or split into sub-goals by invoking previous results, by way of the proof assistant's own *tactics*. For instance, a proof of a statement which holds for every natural number may be chosen to be carried out by induction; if so, the proof assistant will ask for the base case, provide the inductive hypothesis and prompt the user again for the inductive step. To a certain (so far, minimal) extent, the construction of proofs can even be automated.

Libraries for HoTT have been developed on several proof assistants, such as Coq [Coq; Hoq; Bau+17; UniMath], Lean [vDvRB17] and Agda [Agda]. This has lead to the formalization of numerous notions and theorems in various fields of mathematics, most notably in homotopy theory and homological algebra. Lists of formalized results in HoTT appear e.g. in [Uni13, Chapter 8], [vD18] and [Buc20].

In this thesis we focus our attention to category theory, and make use of proof assistants to formulate and formalize in HoTT certain coherence results, which we will now proceed to describe.

1.2 Coherence Theorems in Category Theory

Several notions in category theory stem from algebra. *Monoidal* and *symmetric monoidal categories* are no exception: they originate as the categorification of, respectively, monoids and commutative monoids. In a monoidal category $\mathcal{C}$, the concept of product in a monoid translates to a bi-endofunctor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ which, up to isomorphism, is to respect the monoid axioms of associativity and unitality (and symmetry, if $\mathcal{C}$ is symmetric monoidal).

To support a monoidal product is a feature shared by many categories. Among the most prominent examples, we find the categories of sets (for instance, the cartesian product and the disjoint union); modules over a commutative ring R (direct sum, tensor product over R); small categories (product of categories); and pointed, locally compact topological spaces (coproduct, product, smash product). From a higher categorical perspective, monoidal and symmetric monoidal (1-)categories are shadows of ∞-categories with a product which is parametrized by an A_∞ or E_∞ operad; these, in turn, can be seen as the categorification of spaces equipped with a homotopy-coherently associative (and commutative) multiplication, which are referred to as A_∞- or E_∞-spaces [Ada78] or monoids [e.g. Gep19].

Homotopy-Associative and Homotopy-Commutative Products

A_∞-spaces were introduced by Stasheff [Sta63a; Sta63b] to describe spaces with a homotopy-associative product, and such that its associativity is "homotopy-coherent" regardless of how many terms are involved in such a product.

The prototypical example is given by the loop space ΩX of a pointed topological space $(X, *)$, whose points are loops based at $*$, i.e., continuous pointed maps $p : \mathsf{S}^1 \to X$. A product $\bullet : \Omega X \times \Omega X \to \Omega X$ of loops, also called *concatenation*, can be defined so that the loop $p \bullet q : \mathsf{S}^1 \to X$ runs consecutively through the image of p and that of q, each (for instance) at twice the "speed", as shown in Fig. 1.5.

Associativity of the product is then a homotopy

$$\mathsf{h}_3 : (\Omega X)^3 \times K_3 \to \Omega X \tag{1.1}$$

(depicted in Fig. 1.6), where $K_3 \coloneqq [0,1]$ is the Stasheff polytope of dimension 1, whose vertices are in correspondence to the two ways of concatenating three loops; that is,

$$\mathsf{h}_3(p,q,r,0) = (p \bullet q) \bullet r \qquad \text{and} \qquad \mathsf{h}_3(p,q,r,1) = p \bullet (q \bullet r).$$

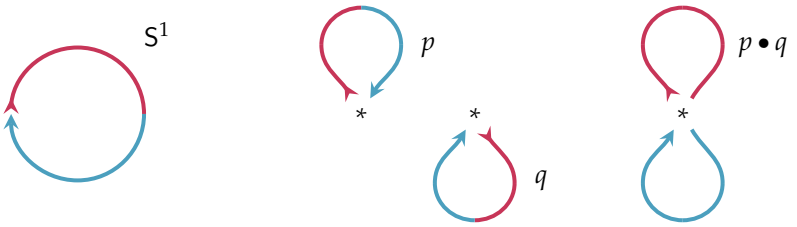


Figure 1.5: Concatenation $p \bullet q$ of loops p and q based at $*$. Using the given colouring of S^1 as a reference, the colouring of the loops suggests how the continuous function $p \bullet q$ is defined, in terms of p and q: the first half of the circle is mapped to the image of p, while the second half is mapped to the image of q.

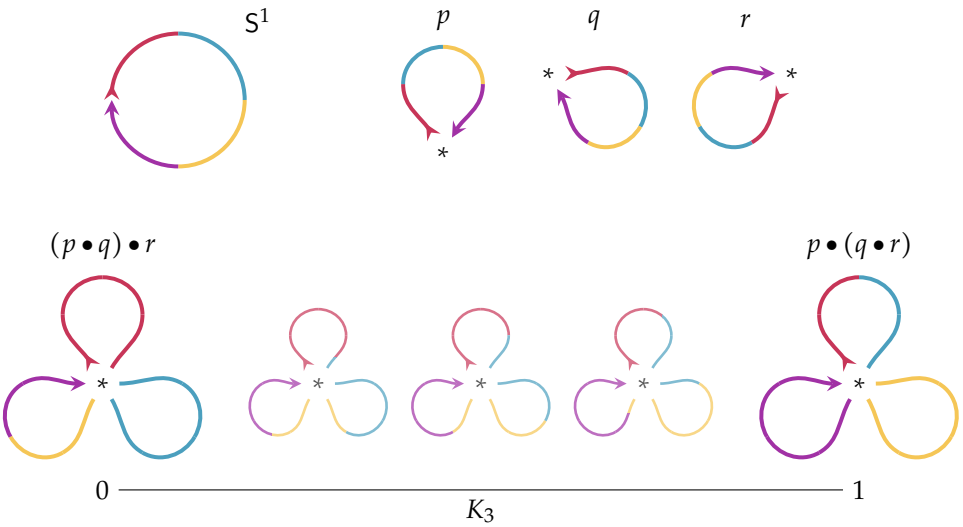


Figure 1.6: Depiction of the homotopy h_3 in (1.1), from the loop $(p \bullet q) \bullet r$ to the loop $p \bullet (q \bullet r)$, witnessing associativity of the concatenation of loops. Again, the colouring is used as a visual aid to mark the difference between the two distinct concatenations of loops.

Analogously, the five ways of concatenating four loops correspond to the vertices of the Stasheff 2-polytope K_4, which has the shape of a filled pentagon, and whose K_3 sides correspond to homotopies h_3; this provides a "homotopy"

$$\mathsf{h}_4 : (\Omega X)^4 \times K_4 \to \Omega X, \tag{1.2}$$

parametrized by K_4, witnessing the first level of coherence of associativity (Fig. 1.7). In general, an n-polytope K_{n+2} is defined for every n, and the ensuing homotopies describe all higher levels of coherence for associativity of the concatenation of any finite number of loops. A_∞-spaces, operads, categories and algebras are discussed in [May72; Ada78; MSS02; Lur17, and many other sources]; some applications are presented in [GJ90].

In order to obviate the inconvenience of carrying homotopies at every level, some *ad hoc* solutions were introduced; the resulting spaces then feature a product that is associative "on the nose", and hence automatically coherent. An example is given by the Moore loop space $\widetilde{\Omega} X$ [see AH56; CM95], of which ΩX is a deformation retract. Moore loops in X based at a point $*$ are pairs (t, p), where $t \in [0, \infty)$

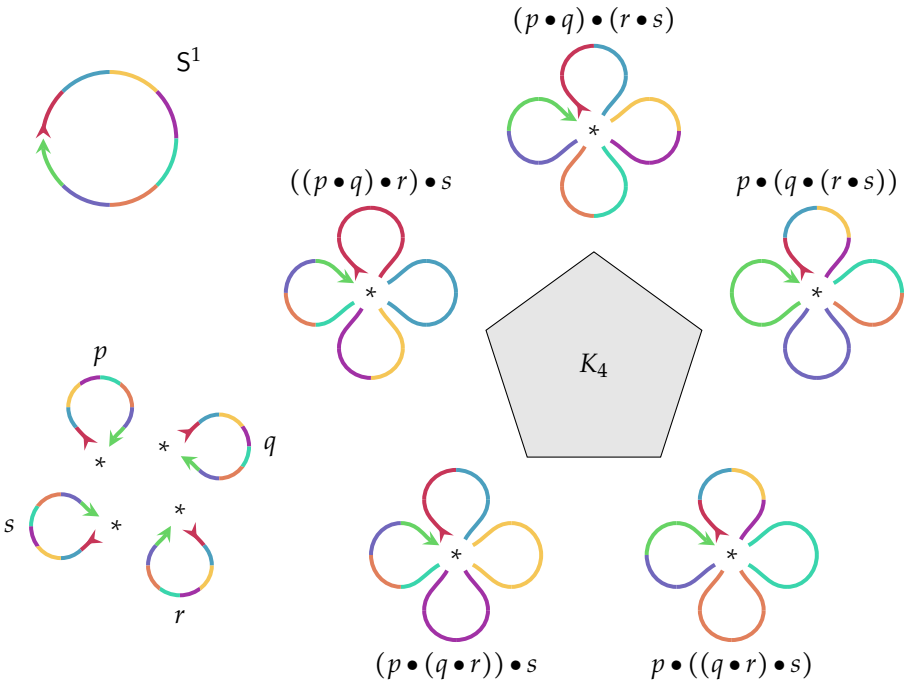


Figure 1.7: Depiction of the homotopy h_4 in (1.2), parametrized by the Stasheff polytope K_4, witnessing the first level of coherence for associativity of the concatenation of loops.

and p is a continuous function $[0,\infty) \to X$ such that $p(0) = * = p(x)$ for every $x \geq t$. One can define a concatenation of Moore loops

$$(t,p) \bullet (s,q) := (t+s, p \cdot q),$$

where $p \cdot q(x)$ agrees with $p(x)$ on arguments $x \leq t$ and with $q(x-t)$ otherwise; this is evidently associative without having to invoke the mediation of a homotopy.

The idea behind E_∞-spaces (-algebras, etc.) is analogous, but deals with symmetry rather than associativity [Ada78]. Again, the loop space construction serves as quintessential example, this time in its iterated version: the product in second loop spaces $\bullet : \Omega^2 X \times \Omega^2 X \to \Omega^2 X$ is homotopy-commutative, i.e., there is a homotopy

$$\mathsf{c}_2 : (\Omega^2 X)^2 \times [0,1] \to \Omega^2 X, \tag{1.3}$$

such that $\mathsf{c}_2(\phi,\psi,0) = \phi \bullet \psi$ and $\mathsf{c}_2(\phi,\psi,1) = \psi \bullet \phi$. This can be obtained via a classical proof known as the "Eckmann-Hilton argument" (originally from [EH62]): indeed, one can show that second loop spaces possess another product $\diamond : \Omega^2 X \times \Omega^2 X \to \Omega^2 X$, satisfying the interchange law

$$(\phi_1 \bullet \phi_2) \diamond (\psi_1 \bullet \psi_2) \sim (\phi_1 \diamond \psi_1) \bullet (\phi_2 \diamond \psi_2). \tag{1.4}$$

Homotopy-commutativity of $\bullet$ is straightforward from (1.4), which moreover implies that the two operations coincide; this is usually explained in terms of "vertical" and "horizontal" compositions, and often proved pictorially – via string diagrams, or by using analogous evocative devices.

Even ignoring the positioning of the parentheses (i.e. assuming that associativity holds strictly, as for Moore loops), the six ways of multiplying three elements of $\Omega^2 X$ in an arbitrary order arrange as the vertices of a hexagonal 2-polytope Z_3 (Fig. 1.8), whose edges correspond to instances of the homotopy c_2. Disappointingly, there is no homotopy parametrized by Z_3 witnessing this sort of coherence for the homotopy-commutativity of the multiplication in a second loop space; there is, however, for the multiplication in a *third* loop space $\Omega^3 X$:

$$\mathsf{c}_3 : (\Omega^3 X)^3 \times Z_3 \to \Omega^3 X. \tag{1.5}$$

In general, higher loop spaces will imply higher levels of coherence. Rather than considering successive iterations of the loop space construction, one is usually interested in those spaces which are (weakly) equivalent to loop spaces of a certain order and can, accordingly, be *delooped* a corresponding number of times. If a space can be delooped as often as desired, we call a choice of such a structure an *infinite loop space*; as it possesses coherence at all levels, it is an E_∞-space. Ω-spectra

are the tool of choice to study infinite loop spaces: for example, the Eilenberg-Mac Lane space $K(G,n)$ of an abelian group G is defined such that its n-th homotopy group $\pi_n(K(G,n))$ is G, while all other homotopy groups are trivial [Whi78, Chapter V]. One can show that $K(G,n)$ is equivalent to $\Omega K(G,n+1)$, which, in turn, is equivalent to $\Omega^2 K(G,n+2)$ – and so forth; thus, Eilenberg-Mac Lane spaces form an Ω-spectrum and, as such, they can be delooped *ad libitum*.

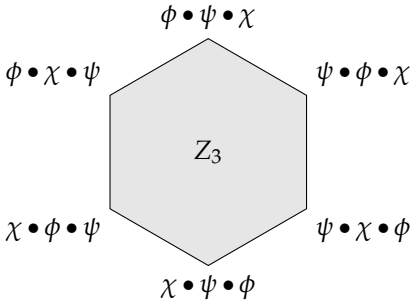


Figure 1.8: Depiction of the polytope Z_3 parametrizing the homotopy c_3 in (1.5), which represents the first level of coherence for the commutativity of the multiplication in an n-fold loop space $\Omega^n X$ for $n \geq 3$ (in which associativity holds strictly). The sides of the hexagon are obtained as instances of the homotopy c_2 in (1.3).

Monoidal Categories and the Coherence Problem

Monoidal, braided monoidal and symmetric monoidal categories address the need for a categorical translation of these structures. In a monoidal category, associativity of the monoidal product $\otimes$ consists of a natural isomorphism $\alpha_{A,B,C} : (A \otimes B) \otimes C \cong A \otimes (B \otimes C)$, which is required to be *coherent*, in the following sense: whenever the five ways of taking the product of four objects arrange in a diagram akin to the one displayed in Fig. 1.7, whose morphisms are obtained as instances of associativity morphisms α, such a diagram commutes. As mentioned, the product in a monoidal category is also unital, i.e., there are natural isomorphisms $\lambda_A : A \otimes E \cong A$ and $\rho_B : E \otimes B \cong B$ where E is a chosen unit object; other relevant "*coherence diagrams*", which we will detail later in this thesis, are also required to commute.

In categories, too, we can ask for associativity (and unitality) of a monoidal product to hold strictly. A monoidal category in which the natural isomorphisms α, λ and ρ are identities is called a *strict* monoidal category. While the categorification process usually implies a relaxation of an algebraic structure up to (a coherent

choice of) homotopy, strict monoidal categories capture the notion of a monoid *before* such a relaxation, since every monoid can already be seen as a discrete strict monoidal category, whose monoidal product is defined by the multiplication in the monoid.

Strict monoidal categories have an easier description than non-strict ("weak") ones: indeed, the coherence diagrams, such as the one evoked by Fig. 1.7, do not play a role in the definition of these categories, since strictness alone enforces their commutativity. Supporting Leinster's suggestive quote at the beginning of this chapter, a theorem of *coherence* for monoidal categories states, essentially, that the two notions coincide: any monoidal category is equivalent to a strict one, via an equivalence that preserves their monoidal structures. A tangible consequence of this theorem – and indeed, an equivalent statement – is that, in a monoidal category, *all* diagrams built out of instances of arrows α, λ and ρ have a decomposition in a patchwork of coherence diagrams for associativity and unitality, and hence they commute. An example is provided in Fig. 1.9.

The notion of a monoidal category can be enhanced to the one of a *braided* monoidal category by supplying it with a natural isomorphism $\tau_{A,B} : A \otimes B \cong B \otimes A$, dubbed "braiding", satisfying certain relations modelled after the definition

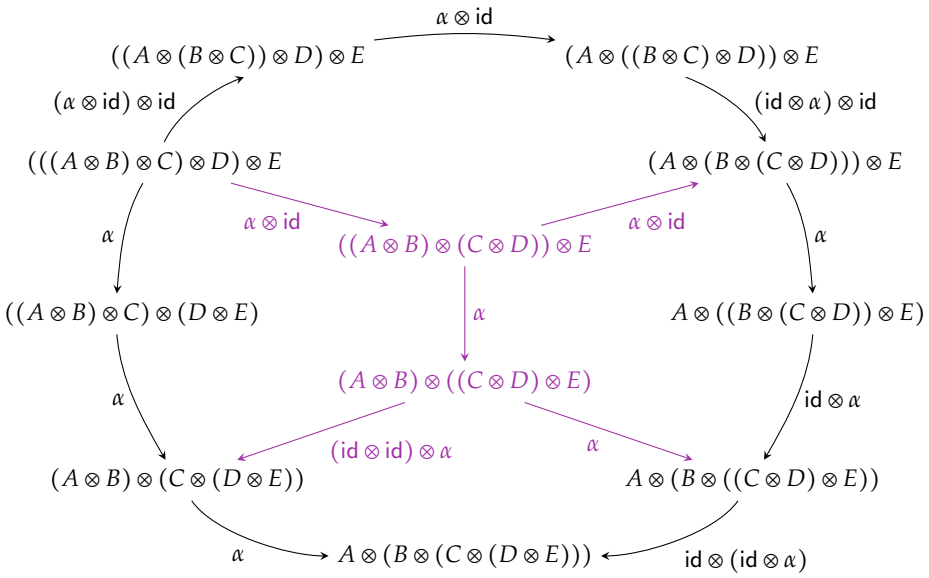


Figure 1.9: Coherence for monoidal categories at work. The outer diagram, built out of instances of the associativity arrows α, can be decomposed in a patchwork of three five-sided coherence diagrams and a four-sided naturality diagram, all of which commute.

of braid groups. The elements of the braid group on n strands describe *braids*, topological objects constructed by twisting n finite-length strings around each other in a 3-dimensional space, without letting them intersect one another or pulling them in the direction opposite to the one into which the braid grows (more precise definitions can be found e.g. in [ML98; KT08]). Multiplication of elements of the group corresponds to joining the respective braids strandwise, while the inverse of a braid is given by letting the braid flow in the opposite direction; a homotopy equivalence between two topological braids reflects an identification between the two elements embodying them in the group (see Fig. 1.10).

If the natural isomorphism τ is its own inverse, then the braiding in the category is governed by symmetric groups rather than braid groups, and the monoidal category is said to be *symmetric*. Essentially, τ and α determine an arrow from any finite product of elements – and any choice of associating them – to the product of any permutation of the same elements. Similarly to those for monoidal categories without a braiding, theorems of coherence for braided monoidal and symmetric monoidal categories have been stated and proved [see e.g. ML98], with the crucial distinction

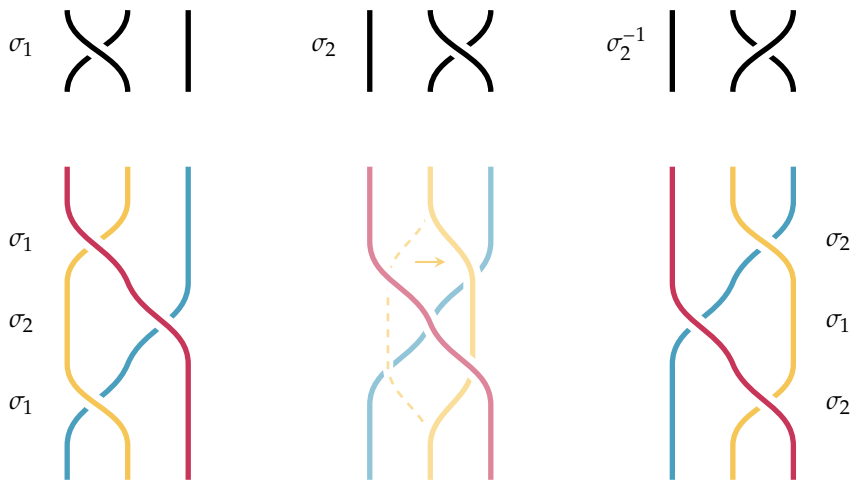


Figure 1.10: Top row: two elementary braids σ_1 and σ_2, corresponding to the generators of the braid group on 3 strands; the inverse σ_2^{-1} is also depicted. Bottom row: the two braids corresponding to the compositions $\sigma_1;\sigma_2;\sigma_1$ (left) and $\sigma_2;\sigma_1;\sigma_2$ (right) are homotopy equivalent; the scheme in the middle shows how to continuously deform the former into the latter, by pulling the central strand from one side to the other. This homotopy equivalence corresponds to the identification of two elements in the braid group, which in this case is also the generating relation in the presentation of the group.

that the classes of commuting diagrams are restricted to those whose arrows are controlled on both sides by the same braiding or permutation. In Fig. 1.11 we show some examples of commuting and noncommuting diagrams in free braided and free symmetric monoidal categories.

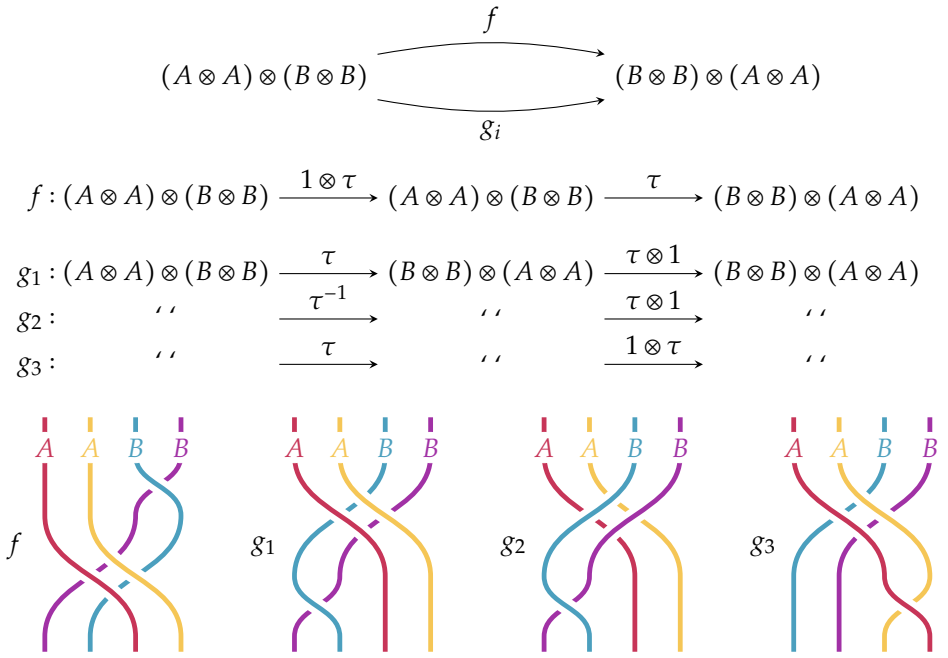


Figure 1.11: Examples of diagrams in free braided and free symmetric monoidal categories, whose commutativity can be established using appropriate coherence theorems. To each of the arrows f, g_1, g_2 and g_3 are associated a braid and a permutation (here expressed by the arrangements of the ends of each braid's strands). For $i = 1$, the relevant permutations coincide and the braids are homotopy equivalent, so the diagram commutes both in free symmetric and in free braided monoidal categories; for $i = 2$, the permutations coincide, but the braids are not homotopy equivalent, so the diagram commutes in free symmetric, but not in free braided, monoidal categories; for $i = 3$, the permutations do not coincide, so the diagram does neither commute in free symmetric nor in free braided monoidal categories.

An interesting class of (symmetric) monoidal categories consists of those which are *closed*. A closed monoidal category $\mathcal{C}$ is equipped with an internal hom functor, i.e., a bifunctor $[-,-] : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{C}$ such that, for every object $A \in \mathcal{C}$, there is an adjunction

$$(- \otimes A) \dashv [A,-] : \mathcal{C} \to \mathcal{C}, \tag{1.6}$$

natural in A. An example is provided by the category of modules over a commuta-

tive ring R, considering the tensor product over R: the internal hom is defined by observing that the set of R-module morphisms possesses itself an R-module structure. In the 1960s, the study of these specific algebraic structures, together with Stasheff's new A_∞-spaces, is reported in [ML76] to have raised a question about *canonicity* of maps, which we proceed to illustrate, and which ultimately instigated the formulation of coherence theorems for monoidal categories.

The tensor product of R-modules is associative: given R-modules M_1, M_2 and M_3, there is an isomorphism $\alpha : (M_1 \otimes_R M_2) \otimes_R M_3 \cong M_1 \otimes_R (M_2 \otimes_R M_3)$, completely determined by declaring

$$\alpha((a_1 \otimes a_2) \otimes a_3) := a_1 \otimes (a_2 \otimes a_3).$$

However, other choices for such an isomorphism are possible. For example, one could adopt an isomorphism α', determined by declaring

$$\alpha'((a_1 \otimes a_2) \otimes a_3) := -a_1 \otimes (a_2 \otimes a_3).$$

What makes α the "canonical" choice for the associativity isomorphism is the fact that it extends to a *general associative law* for any finite number of arguments [Jac51], while α' does not: iteratively applying α' to products of triples of objects in order to obtain an isomorphism

$$((M_1 \otimes_R M_2) \otimes_R M_3) \otimes_R M_4 \cong M_1 \otimes_R (M_2 \otimes_R (M_3 \otimes_R M_4))$$

produces inconsistent results, which depend on the choice of the intermediate steps (expressed exactly by the edges of the Stasheff polytope K_4, as depicted in Fig. 1.7). This inconsistency vanishes when using α instead.

The aptly named *coherence problem*,[1] then, concerns determining the minimal necessary "coherence conditions" to be placed on associativity, unitality $R \otimes_R M \cong M \cong M \otimes_R R$ and symmetry $M_1 \otimes_R M_2 \cong M_2 \otimes_R M_1$ for these isomorphisms to be uniquely determined, in such a way that they extend coherently to isomorphisms between products of any finite number of modules: isomorphisms that we thus consider *canonical*. This problem, of course, generalizes to any other monoidal category (see again Fig. 1.9, exhibiting the coherent behaviour of the associativity isomorphism when extended to a product of five objects). The presence of an internal hom, when considering a closed monoidal category $\mathcal{C}$, opens to a further

[1] The English adjective *coherent* (like the verb *to cohere*) originates from the Latin *con-* (*cum*: "with") + *hærĕo* (*-es, hæsi, hæsum, -ēre*: "to remain unchanged", "to persist"), and it refers to entities having the property of holding together unvaryingly in their different parts.

issue, namely, the ability to find a criterion for establishing the existence of a canonical morphism between two given objects: such are, for example, the internal evaluation morphism $\mathsf{ev}_{A,B} \in \mathcal{C}([A,B] \otimes A,\ B)$, which is left-adjunct to the identity $\mathsf{id} \in \mathcal{C}([A,B],\ [A,B])$ under the adjunction in (1.6), and the internal composition morphism in $\mathcal{C}([B,C] \otimes [A,B],\ [A,C])$, which, under the same adjunction, is right-adjunct to the morphism in $\mathcal{C}(([B,C] \otimes [A,B]) \otimes A,\ C)$ given by the composition $\mathsf{ev}_{B,C} \circ (\mathsf{id} \otimes \mathsf{ev}_{A,B}) \circ \alpha_{[B,C],\,[A,B],\,A}$. In contrast, there are no canonical morphisms in $\mathcal{C}([A,B] \otimes B,\ A)$, nor in $\mathcal{C}([A,C] \otimes [A,B],\ [B,C])$.

A sufficient list of coherence conditions was then identified, leading to the coherence theorems originally presented by Mac Lane [ML63] and by Epstein [Eps66]; one of these conditions was indeed the commutativity of the already described class of five-sided diagrams, now informally known as "Mac Lane's pentagon". Albeit sufficient, the list was later discovered not to be minimal, and it was refined by Kelly [Kel64].

Proofs of Coherence

Nowadays, the coherence theorem for monoidal categories is perhaps best recognized in the formulation appearing in [ML98, Chapter VII]. Mac Lane's proof, which we briefly summarize in Chapter 3, has an intrinsic combinatorial nature: at its core is an argument by induction, appealing to the complexity of the monoidal expressions appearing in a diagram, which is given by the configuration of the parentheses in a product of several objects. The touchstone for the inductive argument is represented by the monoidal expressions which exhibit minimal complexity, i.e., those in which the arrangement of the parentheses prefers one side and with no superfluous instance of the unit object – for example, $A \otimes (B \otimes (C \otimes D))$. These expressions are commonly called *normal forms* and, indeed, we can consider Mac Lane's argument as belonging to a class of proofs "by normalisation", as it unravels the anatomy of the morphisms in a monoidal category by dissecting them into compositions of morphisms leading down to predetermined (normal) expressions. In the same class we can find, for example, Acclavio's result in [Acc17]: there, coherence for monoidal and symmetric monoidal categories is reached using a technique of formal rewriting for string diagrams (introduced in [Laf03]), which are used to represent certain morphisms in such categories.

A different kind of proof was presented by Joyal and Street in [JS86; JS93] and revisited by Leinster in [Lei04]. This proof uses an argument in the style of the Yoneda lemma: any monoidal category $\mathcal{C}$ is shown equivalent to the category of

endofunctors of $\mathcal{C}$ commuting with right translations $(- \otimes B)$; this category is strict monoidal, since the product is given by composition of functors (which is strictly associative and unital). The sought equivalence is achieved via the left translation functor, sending an object $A \in \mathcal{C}$ to the functor $(A \otimes -) : \mathcal{C} \to \mathcal{C}$.

Type-theoretical proofs of coherence, verified by proof assistants, also exist. Notably, Beylin and Dybjer in [BD96; Bey97] make use of a combination of the two proof techniques described above, by using an approach based on *normalisation by evaluation* [see e.g. DF02; AAD07]: monoidal expressions are "interpreted" into functions which produce normal forms when evaluated at some term. The strictification happens, again, precisely because the monoidal product is interpreted into the composition of functions, which is associative and unital on the nose. Since composition of function is not symmetric, it is not immediate how to generalize this technique to prove coherence for symmetric monoidal groupoids; however, the work of Beylin and Dybjer will still serve as a base for this dissertation and will be further discussed in Section 3.7. It is worth mentioning that other formalized proofs have been produced building on the same work [e.g. ABD96].

1.3 Goals and Structure of the Thesis

This monograph is structured as follows.

Chapter 2 includes the concepts and notions in HoTT that are later used in the rest of the thesis.

In Chapter 3 we present a proof of coherence for monoidal groupoids, revisiting Beylin and Dybjer's formalization [BD96; Bey97] by employing distinctive features of HoTT. We exploit the higher groupoid structure of types to define, for a 0-type (set) X, the free monoidal groupoid $\mathsf{FMG}(X)$ generated by X as a HIT; the type is designed so that its elimination principle contains the proof of freeness. Coherence for monoidal groupoids is then achieved by showing that $\mathsf{FMG}(X)$ is, itself, a set: this is reached via a proof of normalisation of the terms in $\mathsf{FMG}(X)$ into the type $\mathsf{list}(X)$ of lists over X. We also offer a comparison of our work to other known proofs of coherence, of which we give a brief account.

In Chapter 4 we extend the same result to obtain a formalized technique of normalisation for symmetric monoidal expressions into unordered lists. In this case, both the type $\mathsf{FSMG}(X)$ of free symmetric monoidal expressions and the type $\mathsf{slist}(X)$ of unordered lists are defined as HITs, and neither of them is a set.

In Chapter 5 we further investigate symmetric monoidal structures, examining the connection between free symmetric monoidal groupoids and finite types. We

propose a strategy to prove that the subuniverse $\mathcal{B}\mathsf{S}_\bullet$ of finite types, which represents the classifying space of symmetric groups, is a free symmetric monoidal groupoid. In order to do so, we try to construct a chain of symmetric monoidal equivalences

$$\mathsf{slist}(\mathbf{1}) \simeq \mathsf{del}_\bullet \simeq \mathcal{B}\mathsf{S}_\bullet,$$

where $\mathbf{1}$ is the unit type, and $\mathsf{del}_\bullet$ is the type of deloopings of symmetric groups, defined as a family of HITs indexed by the natural numbers. While the leftmost symmetric monoidal equivalence is fully formalized, the proof of equivalence between $\mathsf{del}_\bullet$ and $\mathcal{B}\mathsf{S}_\bullet$ relies on a few unformalized statements. Assuming the latter, we are able to easily isolate, in a free symmetric monoidal groupoids, the class of diagrams involving symmetric monoidal expressions without repetitions, and to prove that all those diagrams commute.

In Chapter 6 we discuss possible research trajectories and alternative formulations of the statements of coherence for monoidal and symmetric monoidal groupoids.

While the main results presented in this text are the formalization of mathematical objects such as monoidal and symmetric monoidal structures and the production of computer-verified proofs of coherence, the implicit objectives of this thesis, both practical and theoretical, are manifold. Firstly, we aim to discover how to exploit the peculiarities of HoTT in order to produce proofs that are short, elegant and adaptable to different frameworks. At the same time, in the opposite direction, we will investigate some of the constraints given by the expressivity of the theory, which will render certain notions harder or very impractical to formalize. In addition, and not less importantly, we want to assess the feasibility of proof verification in HoTT – using the proof assistant Coq – on the subject of category theory. All of this will force us to consider several choices for stating and proving coherence; by weighing those against each other, we will highlight the features of our approach with respect to other known proofs, both formalized and not.

A large part of the work we present has been verified using the HoTT library for the proof assistant Coq [Hoq]. The latest version of the formalization prior to the submission of this thesis is to be considered supplementary material, part of which is presented and discussed in Appendix A.

Chapter 2

Homotopy Type Theory

In this chapter we will highlight selected ideas and definitions in HoTT, while fixing the notation for terms and types that we will employ throughout this thesis. The main reference is [Uni13], a textbook laying the basis for Homotopy Type Theory and Univalent Foundations, which gave rise to the expression "book HoTT" to refer to the type theory described therein.

2.1 Types, Terms and Judgments

A **type** is a primitive concept of HoTT and of any type theory, like a set is a primitive concept of set theory. As for sets in the Zermelo-Fraenkel axiomatization of set theory [Fra73; see also Lei14], types are then not subject to a definition of the kind "a type *is*..."; rather, each of them comes with a collection of *rules* which specify how to work with them. Another primitive concept is that of **term**, which is a *typed* syntactic expression (string of text); every term has, necessarily, a type. We will frequently use capital letters to denote types and lower-case letters to denote terms. That a term x is of type X is denoted by the string

$$x : X$$

which, itself, belongs to a class of expressions that goes under the name of **judgments** of the type theory.

Some of the rules of type theory are called **introduction rules**; they indicate how to produce a term of a given type, by declaring judgments which stipulate that some term expressions are of a certain type. Syntactic expressions can be matched against those provided by the introduction rules, both to verify that they are well-typed and to define functions recursively by case analysis. We also have *elimination rules*, which specify how to use a term in some type to derive others; the duality introduction/elimination is expressed by means of *computation rules* (see Remark 2.11).

As mentioned in the introduction of this thesis, the kind of mathematics allowed in HoTT is inherently constructive: a proof of a statement of the form "there exists an object x satisfying a property P" needs to be presented as a term of a certain type (a "Σ-type", presented in Section 2.2), which will have to make explicit the instance x. Proving a result, then, corresponds to giving a term in the type representing its statement; in this sense, the line between a definition and a theorem becomes blurred. Hence, throughout this thesis, theorems and lemmata are to be considered special cases of definitions.

An important class of types in HoTT (and MLTT) is the class of function types (Section 2.2), the terms of which are *functions* from a domain to a target type. A function can be applied to terms of its domain, producing a term of the target type. In many cases, the definition of a function will make it *compute* when applied to certain terms; for example, a function double defined on the type of natural numbers (Section 2.3) can be defined so that it computes to the term 6 when applied to the term 3. Computation of terms rests on concepts of *reduction*, *substitution* and *conversion* of expressions; without the need of making this description more explicit in the context of this thesis, we will just denote two term expressions x and y of the same type with the string (judgment)

$$x \equiv y \qquad\qquad (\text{e.g. } \mathsf{double}(3) \equiv 6)$$

if they compute to the same term. The relation $\equiv$, called **judgmental equality**, is an equivalence relation; two judgmentally equal terms are interchangeable in every expression that contains them. We will also use the notation

$$x :\equiv y \qquad\qquad (\text{e.g. } t :\equiv \mathsf{double}(3))$$

to define terms by assigning the computational content of y to x. We will liberally combine this notation, e.g. "$t :\equiv \mathsf{double}(3) \equiv 6 : \mathbb{N}$" means that the term t is defined as $\mathsf{double}(3)$, which is judgmentally equal to 6, and all of them have type $\mathbb{N}$.

Every type is a term of some **universe** (Section 2.5), which is, itself, a type. Every universe type is a term of another universe, higher up in a hierarchy of universes. As we will never be concerned with the specific universe a type belongs to, we will denote all universes uniformly as $\mathcal{U}$.

HoTT is a *dependent* type theory: as such, it allows definitions of **families** of types indexed over the terms of a type. Given a type $X : \mathcal{U}$, a family $Y : X \to \mathcal{U}$ has, as members, types $Y(x)$ for every $x : X$. If $Y(x) \equiv Z$ for every x and for some type Z, with x not occurring in Z, the family is said to be *constant*; every type can

be then considered as a constant family of types over any other type. This expands the concept of *functions* between types to that of *dependent functions*; both will be described in the next section.

A guiding principle when working in HoTT is to interpret types as spaces, terms as points, functions as continuous functions, families as fibrations and dependent functions as sections (Remark 2.56). As hinted in the introduction, certain types, named *identity types*, can be then interpreted as path spaces (Section 2.4).

We will begin by describing some of the types available to us in HoTT.

2.2 Functions and Pairs

Π-Types

Definition 2.1 (Function types)**.** Given a type A and a family $B : A \to \mathcal{U}$, there is a type, called *(dependent) function type* or **Π-type**, denoted by

$$\Pi\,(x : A)\,.\,B(x).$$

Its terms are called *(dependent) functions* from A to the family B. An expression of the form

$$(x \mapsto b(x)) \tag{2.2}$$

is a term of $\Pi\,(x : A)\,.\,B(x)$ whenever $b(x)$ is a term of type $B(x)$ for every $x : A$. A function term $f : \Pi\,(x : A)\,.\,B(x)$ can be *applied* to a term $a : A$; the application is denoted by $f(a) : B(a)$. The following computation rules hold:

$$(x \mapsto b(x))(a) \equiv b(a) \tag{2.3}$$

$$(x \mapsto f(x)) \equiv f. \tag{2.4}$$

If B is a constant family (i.e., just a type), the same type is denoted by $A \to B$ and its terms are called *(non-dependent) functions* from A to B.

Remark 2.5 (Notation). The introduction rule (2.2) is known as *lambda-abstraction*, and indeed function terms $(a \mapsto b(a))$ are universally denoted by $\lambda a.b(a)$. In this thesis, we will use the lowercase Greek letter λ to denote the left-unitality arrow in a monoidal category, so we choose a different notation for function terms to avoid confusion. The judgmental equalities in the computation rules (2.3)–(2.4) are known, respectively, as *beta-reduction* (when applied from left to right) and *eta-conversion*.

Function type expressions associate to the right: the notation $A \to B \to C$ indicates the type $A \to (B \to C)$. If f is a function in such a type, we will often shorten a term $f(a)(b) : C$ as $f(a,b)$ for $a : A$ and $b : B$; this will also hold for dependent functions $f : \Pi\,(a : A)\,.\,\Pi\,(b : B(a))\,.\,C(a,b)$. In such a situation, we may sometimes omit some arguments of the function for brevity (e.g. writing $f(b)$ instead of $f(a,b)$) or make use of subscripts (as in $f_a(b)$).

Definition 2.6. Given a type X, the *identity function* $\mathsf{id}_X : X \to X$ is defined as

$$\mathsf{id}_X :\equiv (x \mapsto x).$$

The function id_X will be denoted by id when the type X is clear from the context. Given types X, Y and Z and functions $f : X \to Y$ and $g : Y \to Z$, the *composition* $g \circ f$ is defined as

$$g \circ f :\equiv (x \mapsto g(f(x))).$$

Remark 2.7. Consistently with an understanding of types and functions as objects and arrows of a category, the definitions of identity and composition make it so that composition of functions is *judgmentally* associative and respects unit laws. Indeed, given functions $f : W \to X$, $g : X \to Y$ and $h : Y \to Z$, we have the judgmental equality:

$$(h \circ g) \circ f \equiv h \circ (g \circ f);$$

given a function $f : X \to Y$, we have:

$$f \circ \mathsf{id}_X \equiv f \equiv \mathsf{id}_Y \circ f.$$

The judgmental equalities are here obtained via beta-reduction and eta-conversion; for example, the right unit law is achieved by means of the following chain of judgmental equalities:

$$\begin{aligned}
f \circ \mathsf{id}_X &\equiv (x \mapsto f(\mathsf{id}_X(x))) && \text{by Definition 2.6}\\
&\equiv (x \mapsto f((y \mapsto y)(x))) && \text{by Definition 2.6}\\
&\equiv (x \mapsto f(x)) && \text{by (2.3)}\\
&\equiv f && \text{by (2.4).}
\end{aligned}$$

Σ-Types

Definition 2.8 (Σ-types)**.** Given a type A and a family $B : A \to \mathcal{U}$, there is a type, called *(dependent) pair type* or **Σ-type**, denoted by

$$\Sigma\,(x : A)\,.\,B(x).$$

Its introduction rule states that an expression of the form $\langle x, b(x)\rangle$ is a term of $\Sigma\,(x : A)\,.\,B(x)$ whenever $x : A$ and $b(x) : B(x)$; such term expressions are called *(dependent) pairs*. Given a family $C : (\Sigma\,(x : A)\,.\,B(x)) \to \mathcal{U}$ and a dependent function $f : \Pi\,(a : A)\,.\,\Pi\,(b : B(a))\,.\,C\langle a, b\rangle$, there is a function

$$\mathsf{ind}_\Sigma : \Pi\,(p : \Sigma\,(x : A)\,.\,B(x))\,.\,C(p) \tag{2.9}$$

satisfying the computation rule

$$\mathsf{ind}_\Sigma\langle a, b\rangle \equiv f\langle a, b\rangle. \tag{2.10}$$

If B is a constant family, the same type is denoted by $A \times B$ and is called a *product type*.

From Definition 2.8, we see that, *a priori*, a term of a Σ-type is not necessarily a pair, in the sense that a term variable of a Σ-type cannot be made to judgmentally reduce to a pair. However, once we define a notion of term identity, we will see that every term in a Σ-type can be, in some precise sense, identified with a pair (Lemma 2.47).

Remark 2.11. The term ind_Σ in (2.9) is the **elimination principle** or **rule** of the Σ-type, while the judgmental equality in (2.10) is its **computation rule**. Together, elimination rules and their computation rules can be seen as universal properties of types. Elimination rules are also called *induction principles* in their more general form, or sometimes *recursion principles* if the source or target family of the universal property is constant (in the definition above, the family C). The universal property of Σ-types then states that, in order to define a map out of a Σ-type, it is enough to define its behaviour on pairs. In general, we will say that we eliminate *into* the target type or family of the elimination principle of a certain type (e.g., the family C in (2.9)).

Remark 2.12. When instantiated to a product type $A \times B$ and to a constant family C, the elimination principle for product types states that a function $A \times B \to C$ can be produced once given a function $A \to B \to C$. This is also known as *uncurrying*, where *currying* is the reverse operation; these two function types are indeed *equivalent* (in the sense of Section 2.5). There is a clear analogy with Hom functors in different contexts (e.g. closed monoidal categories), for which there is a bijection

$$\mathrm{Hom}(X \times Y, Z) \leftrightarrow \mathrm{Hom}(X, Z^Y),$$

natural in X and Z. This justifies the use of notation described in Remark 2.5 ($f(a, b)$ vs. $f(a)(b)$); the different notations $\langle a, b\rangle$ and (a, b) for terms in a product type and lists of arguments of a function should clarify the context.

Remark 2.13. It is worth to remark that, when we present a term under the assumption that some data is given, we are actually specifying a dependent function (depending, precisely, on the given data). In its complete form, the elimination principle for Σ-types in (2.9) postulates the existence of a term

$$\begin{aligned}\mathsf{ind}_\Sigma :{}& \Pi\,(A:\mathcal{U})\,.\,\Pi\,(B:A\to\mathcal{U})\,.\,\Pi\,(C:(\Sigma\,(x:A)\,.\,B(x))\to\mathcal{U})\,.\\ &(\Pi\,(a:A)\,.\,\Pi\,(b:B(a))\,.\,C\langle a,b\rangle)\to\Pi\,(p:\Sigma\,(x:A)\,.\,B(x))\,.\,C(p)\end{aligned}\tag{2.14}$$

satisfying

$$\mathsf{ind}_\Sigma(A,B,C,f)\langle a,b\rangle \equiv f\langle a,b\rangle$$

for every A, B, C, f, a and b of appropriate types.

Remark 2.15 (Notation). We emphasize that the expression "a" in $\Sigma\,(a:A)\,.\,B(a)$ is a bound variable, so it bears no significance in the definition of the type; the same type can be written as $\Sigma\,(x:A)\,.\,B(x)$.

Tuples will denote iterated pairs: for appropriate A, B and C, a term $\langle a,\langle b,c\rangle\rangle : \Sigma\,(a:A)\,.\,\Sigma\,(b:B(a))\,.\,C(a,b)$ will be written as $\langle a,b,c\rangle$.

The operator $\times$ has precedence over $\to$, so the type $A\to B\times C$ is to be read as $A\to(B\times C)$, while $A\times B\to C$ denotes $(A\times B)\to C$.

The elimination principle for Σ-types allows us to define *projections*, which will also serve as our first example of a definition by induction.

Definition 2.16. Given a type A and a family $B:A\to\mathcal{U}$, the functions

$$\begin{aligned}&\mathsf{pr}_1 : (\Sigma\,(a:A)\,.\,B(a))\to A \quad \text{and}\\ &\mathsf{pr}_2 : \Pi\,(x:\Sigma\,(a:A)\,.\,B(a))\,.\,B(\mathsf{pr}_1(x))\end{aligned}$$

(respectively, first and second projection) are defined *by induction*, so that $\mathsf{pr}_1\langle a,b\rangle :\equiv a$ and $\mathsf{pr}_2\langle a,b\rangle :\equiv b$ for every $a:A$ and $b:B(a)$. We will use the same notation when B is a constant family.

The expression "*by induction*" above is to be read as the definition of the terms:

$$\begin{aligned}&\mathsf{pr}_1 :\equiv \mathsf{ind}_\Sigma(A,B,(z\mapsto A),(a\mapsto(b\mapsto a))),\\ &\mathsf{pr}_2 :\equiv \mathsf{ind}_\Sigma(A,B,(z\mapsto B(\mathsf{pr}_1(z))),(a\mapsto(b\mapsto b))),\end{aligned}$$

where all the arguments of ind_Σ are explicit (as in (2.14)). Most definitions in this thesis will follow the same pattern.

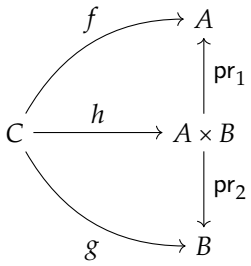


Figure 2.1: Universal property of pair types.

Remark 2.17. Given types A, B and C and functions $f : C \to A$ and $g : C \to B$, we can construct a map $h :\equiv (c \mapsto \langle f(c), g(c)\rangle) : C \to A \times B$ such that $\mathsf{pr}_1 \circ h \equiv f$ and $\mathsf{pr}_2 \circ h \equiv g$ (Fig. 2.1).

The following notion of functoriality holds for Σ-types.

Definition 2.18. Let A, A' be types and $B : A \to \mathcal{U}$, $B' : A' \to \mathcal{U}$ be families. Given functions $f : A \to A'$ and $g : \Pi\,(x : A)\,.\,B(x) \to B'(f(x))$, the function

$$\langle f, g\rangle : \Sigma\,(x : A)\,.\,B(x) \to \Sigma\left(y : A'\right).\,B'(y)$$

is defined by induction, so that $\langle f, g\rangle\langle a, b\rangle :\equiv \langle f(a), g_a(b)\rangle$. If B and B' are constant and $g : B \to B'$, we denote by $f \times g$ the function between product types:

$$f \times g :\equiv \langle f, (a \mapsto g)\rangle : A \times B \to A' \times B'.$$

We can interpret Σ- and Π-types as, respectively, the left adjoint and the right adjoint to a constant functor, in the following sense. Given a type A, there are functions:

- $\Sigma_A : (A \to \mathcal{U}) \to \mathcal{U}$, defined as $\Sigma_A :\equiv (B \mapsto \Sigma\,(a : A)\,.\,B(a))$;
- $\Pi_A : (A \to \mathcal{U}) \to \mathcal{U}$, defined as $\Pi_A :\equiv (B \mapsto \Pi\,(a : A)\,.\,B(a))$;
- $\mathrm{Const}^A : \mathcal{U} \to (A \to \mathcal{U})$, defined as $\mathrm{Const}^A :\equiv (X \mapsto (a \mapsto X))$.

Viewing the universe $\mathcal{U}$ as a category of types and functions (setting aside the issues that this interpretation might give),[1] the type $A \to \mathcal{U}$ corresponds to the coslice category $A{\downarrow}\mathcal{U}$, whose morphisms $(A{\downarrow}\mathcal{U})(B, B')$ are dependent functions $\Pi\,(x : A)\,.\,B(x) \to B'(x)$. In this sense, Σ_A, Π_A and Const_A are actually functors: we can define function terms in

$$(A{\downarrow}\mathcal{U})(B, B') \to (\Sigma_A(B) \to \Sigma_A(B')) \quad \text{and} \quad (A{\downarrow}\mathcal{U})(B, B') \to (\Pi_A(B) \to \Pi_A(B'))$$

[1] These are sometimes called *wild* categories.

for B, $B' : A \to \mathcal{U}$, and a function term in

$$(B \to B') \to (A \downarrow \mathcal{U})(\mathrm{Const}^A(B), \mathrm{Const}^A(B'))$$

for B, $B' : \mathcal{U}$, judgmentally respecting identity and composition. Such functions present the functorial action of Σ_A, Π_A and Const^A on maps, and are defined as follows:

- for Σ_A, given a morphism $f : (A \downarrow \mathcal{U})(B, B')$, we can produce

  $$\langle \mathsf{id}_A, f \rangle : \Sigma\,(x : A)\,.\,B(x) \to \Sigma\,(x : A)\,.\,B'(x)$$

  as shown in Definition 2.18;
- for Π_A, given a morphism $f : (A \downarrow \mathcal{U})(B, B')$, we can produce the (dependent) composition

  $$(h \mapsto (z \mapsto f(z) \circ h(z))) : (\Pi\,(x : A)\,.\,B(x)) \to \Pi\,(x : A)\,.\,B'(x);$$

- for Const_A, given a morphism $g : X \to X'$, we can produce

  $$(a \mapsto g) : A \to X \to X'.$$

These functors are in an adjoint triple

$$\Sigma_A \dashv \mathrm{Const}^A \dashv \Pi_A \;:\; A \downarrow \mathcal{U} \to \mathcal{U},$$

which gives rise to two adjunctions:

$$\Sigma_A \,\mathrm{Const}^A \dashv \Pi_A \,\mathrm{Const}^A \;:\; \mathcal{U} \to \mathcal{U}, \tag{2.19}$$

$$\mathrm{Const}^A \,\Sigma_A \dashv \mathrm{Const}^A \,\Pi_A \;:\; A \downarrow \mathcal{U} \to A \downarrow \mathcal{U}. \tag{2.20}$$

By computing the compositions, the adjunction in (2.19) corresponds, categorically, to the adjunction $(A \times -) \dashv (A \to -)$ (cf. Remark 2.12), while the one in (2.20) is its "dependent" counterpart $(\Sigma\,(x : A)\,.\,-(x)) \dashv (\Pi\,(x : A)\,.\,-(x))$.

2.3 Inductive Types

We saw in Definition 2.16 and Definition 2.18 how the elimination principle of Σ-types allows us to define functions out of a Σ-type by declaring the image of the terms specified by its introduction rule, i.e. pair terms; indeed, the type itself is defined by means of this universal property. Types that are defined in terms of such a

universal property are called **inductive types**. The presentation of inductive types is given by a list of *constructors* (also called *generators*) of their terms, specifying the introduction rule(s), and by an elimination principle, which can often be inferred by the list of constructors alone. This section presents inductive types that are commonly used.

The Unit Type, the Empty Type, and the Type of Booleans

Definition 2.21 (Unit type)**.** The inductive type $\mathbf{1}$ has one constructor, denoted by the symbol $* : \mathbf{1}$. Given a family $C : \mathbf{1} \to \mathcal{U}$ and a term $c : C(*)$, there is a function

$$\mathsf{ind}_{\mathbf{1}} : (x : \mathbf{1}) \to C(x)$$

satisfying the computation rule $\mathsf{ind}_{\mathbf{1}}(*) \equiv c$. The type $\mathbf{1}$ is called the **unit type**.

Definition 2.22 (Empty type)**.** The inductive type $\mathbf{0}$ has no introduction rule, and hence it is called the **empty type**. Given a family $C : \mathbf{0} \to \mathcal{U}$, there is a function

$$\mathsf{ind}_{\mathbf{0}} : \Pi\,(x : \mathbf{0})\,.\,C(x),$$

also called *ex falso*: it states that a term of any type can be obtained, given a term in $\mathbf{0}$. The elimination principle cannot be applied to any constructor of $\mathbf{0}$ (because there is none), so no computation rule applies.

The unit type is a "type with exactly one term": its elimination principle will be used to show (in Lemma 2.47) that all terms in $\mathbf{1}$ can be brought back to $*$; hence, this type will be used as building block to construct types with finitely many elements (Definition 2.43). From a homotopical perspective, $\mathbf{1}$ will be a contractible type (Remark 2.75). Similarly, the empty type is a "type with no terms", as no term of $\mathbf{0}$ can be produced. If a function $f : X \to \mathbf{0}$ is exhibited, the type X is itself empty (see also Lemma 2.96); if X represents a mathematical statement, it has no proof.

The type $\mathbf{1}$ is terminal in $\mathcal{U}$: given any type X, a function $\mathsf{const}_* : X \to \mathbf{1}$ can be always defined as the constant map at $*$. Dually, the elimination principle of $\mathbf{0}$ shows that the empty type is initial in $\mathcal{U}$.

Definition 2.23 (Type of booleans)**.** The inductive type $\mathbf{2}$ has two constructors, yes and $\mathsf{no} : \mathbf{2}$. Given a family $C : \mathbf{2} \to \mathcal{U}$ and terms $y : C(\mathsf{yes})$ and $n : C(\mathsf{no})$, there is a function

$$\mathsf{ind}_{\mathbf{2}} : \Pi\,(x : \mathbf{2})\,.\,C(x)$$

satisfying the computation rules $\mathsf{ind}_{\mathbf{2}}(\mathsf{yes}) \equiv y$ and $\mathsf{ind}_{\mathbf{2}}(\mathsf{no}) \equiv n$.

Natural Numbers and Lists

The definition of inductive types allows *recursion*: one or more of the constructors of an inductive type might quantify over the defined type itself. Recursive definitions are common in mathematics; here we present the inductive types of natural numbers and of lists (over a type), which we will use in several occasions throughout this thesis.

Definition 2.24 (Natural numbers)**.** The inductive type $\mathbb{N}$ of **natural numbers** has two constructors, $0 : \mathbb{N}$ and $\mathsf{s} : \mathbb{N} \to \mathbb{N}$, respectively for the "zero" and the successor function in the natural numbers. Given a family $C : \mathbb{N} \to \mathcal{U}$, a term $z : C(0)$ and a function $f : \Pi\,(n : \mathbb{N})\,.\,C(n) \to C(\mathsf{s}(n))$, there is a function

$$\mathsf{ind}_{\mathbb{N}} : \Pi\,(n : \mathbb{N})\,.\,C(n) \tag{2.25}$$

satisfying the computation rules

$$\begin{aligned} \mathsf{ind}_{\mathbb{N}}(0) &\equiv z \\ \mathsf{ind}_{\mathbb{N}}(\mathsf{s}(n)) &\equiv f(n, \mathsf{ind}_{\mathbb{N}}(n)) \end{aligned} \tag{2.26}$$

where the judgmental equality in (2.26) holds for every $n : \mathbb{N}$. This elimination principle gives computational content to the classical induction on the natural numbers: the right-hand side in (2.26) is the inductive step in a proof by induction on $\mathbb{N}$, which recursively calls ind_N as proof of the induction hypothesis (given by the second argument of f).

Remark 2.27 (Notation). We will use the usual notation for natural numbers (0, 1, 2, ...) as shorthand for terms in $\mathbb{N}$. The operation of addition $+ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ on natural numbers can be defined by induction either on the left or on the right summand (producing functions with different computational behaviour); in this thesis, we will choose the *former*, but often denote $1 + n \equiv \mathsf{s}(n)$ by "$n + 1$", purely for reasons of improved readability.

Remark 2.28. The elimination rule for $\mathbb{N}$ has a simpler formulation when C is a constant family. It states that, given a type C, a term $z : C$ and a function $f : \mathbb{N} \to C \to C$, there is a function $\mathsf{rec}_{\mathbb{N}} : \mathbb{N} \to C$, satisfying computation rules corresponding to those of ind_N. The non-dependent version rec of an elimination principle can always be derived from its dependent version ind; hence, we will often omit its specification and use rec whenever we wish to emphasize its non-dependent character (see also the notes at the end of this section).

Definition 2.29 (Lists)**.** Given a type X, the inductive type $\mathsf{list}(X)$ of **lists** over X has two constructors: $\mathsf{nil} : \mathsf{list}(X)$, the "empty list"; and $\mathsf{cons} : X \to \mathsf{list}(X) \to \mathsf{list}(X)$, which constructs a new list with a head in X and a tail in $\mathsf{list}(X)$. Given a family $C : \mathsf{list}(X) \to \mathcal{U}$, a term $e : C(\mathsf{nil})$ and a function $c : \Pi\,(x : X)\,.\,\Pi\,(l : \mathsf{list}(X))\,.\,C(l) \to C(\mathsf{cons}(x, l))$, there is a function

$$\mathsf{ind}_{\mathsf{list}} : \Pi\,(l : \mathsf{list}(X))\,.\,C(l)$$

satisfying the computation rules

$$\begin{aligned} \mathsf{ind}_{\mathsf{list}}(\mathsf{nil}) &\equiv e \\ \mathsf{ind}_{\mathsf{list}}(\mathsf{cons}(x, l)) &\equiv c(x, l, \mathsf{ind}_{\mathsf{list}}(l)) \end{aligned} \tag{2.30}$$

where the judgmental equality in (2.30) holds for every $x : X$ and $l : \mathsf{list}(X)$.

Remark 2.31 (Notation). Whenever we apply the constructor cons to two arguments, we replace the notation $\mathsf{cons}(x, l)$ by $x :: l$ (obviously associating to the right), which has the advantage of displaying a list term graphically as a list. For instance, if a, b, $c : X$, the expression $a :: b :: c :: a :: \mathsf{nil}$ is a term of $\mathsf{list}(X)$. We will still keep the notation cons in some cases, especially when the function is not completely applied.

Lists will be of particular importance in this thesis: they possess a monoidal structure, with nil as unit. We will use the elimination rule to define its monoidal product, which is the operation of appending lists. The elimination rule states that, in order to define a function out of the type of lists, it is enough to give the image of nil and, recursively, the image of $x :: l$ for every x, given the image of some $l : \mathsf{list}(X)$ as inductive hypothesis.

Definition 2.32 (List append)**.** Given a type X, the function $- +\!\!+ - : \mathsf{list}(X) \to \mathsf{list}(X) \to \mathsf{list}(X)$ is defined by induction (on its first argument) by:

$$(\mathsf{nil} +\!\!+ -) :\equiv \mathsf{id}_{\mathsf{list}(X)} \tag{2.33}$$

$$((x :: l) +\!\!+ -) :\equiv \mathsf{cons}(x) \circ (l +\!\!+ -) \tag{2.34}$$

for every $x : X$ and $l : \mathsf{list}(X)$.

Remark 2.35 (Notation). We use the convention that makes $+\!\!+$ associate to the right, so $l_1 +\!\!+ l_2 +\!\!+ l_3$ stands for $l_1 +\!\!+ (l_2 +\!\!+ l_3)$. Clearly, $::$ has priority on the right of $+\!\!+$, as there is only one way of reading an expression such as $a +\!\!+ b :: c$ which makes it type-check. On the left, it does not matter, as (2.34) implies that the expressions $(x :: l_1) +\!\!+ l_2$ and $x :: (l_1 +\!\!+ l_2)$ are judgmentally equal.

Remark 2.36. The operation $+\!+$ satisfies the left-unit law judgmentally, as (2.33) implies that $\mathsf{nil} +\!+ l \equiv l$ for all $l : \mathsf{list}(X)$. Instances of $(- +\!+ -)$ applied to nil as second argument and any concrete list (for example, the one in Remark 2.31) as first argument do compute to the first argument; however, there is no computation rule stating that $l +\!+ \mathsf{nil} \equiv l$ given *any* term $l : \mathsf{list}(X)$. Indeed, all we know about $(l +\!+ -)$ is that it is a function $\mathsf{list}(X) \to \mathsf{list}(X)$, and we do not have means to compute its value when applied to nil (or any other list). We will see in Chapter 3 how identity types, describing a weaker notion of equality, will allow us to express the right-unit law for $+\!+$.

Coproduct Types and Canonical Finite Types

The types $\mathbf{0}$, $\mathbf{1}$ and $\mathbf{2}$ are inductive types with zero, one and two constructors respectively; it is not difficult to imagine similar definitions for types with an arbitrary (but fixed) number of constructors. The types $\mathsf{list}(X)$ and $\mathbb{N}$ offer examples of inductive types with *families* of constructors (respectively: cons and s, which also make the definitions recursive). Coproduct types are another such example.

Definition 2.37 (Coproduct type)**.** Given types A and B, the inductive type $A + B$, called **coproduct type** of A and B, has two families of constructors:

$$\mathsf{inl} : A \to A + B \qquad \text{and} \qquad \mathsf{inr} : B \to A + B.$$

Given a family $C : A + B \to \mathcal{U}$ and dependent functions $f : \Pi\,(a : A)\,.\,C(\mathsf{inl}(a))$ and $g : \Pi\,(b : B)\,.\,C(\mathsf{inr}(b))$, there is a function

$$\mathsf{ind}_+ : \Pi\,(x : A + B)\,.\,C(x)$$

satisfying the computation rules $\mathsf{ind}_+(\mathsf{inl}(a)) \equiv f(a)$ and $\mathsf{ind}_+(\mathsf{inr}(b)) \equiv g(b)$ for every $a : A$ and $b : B$. Thus, when proving a statement depending on a term $x : A + B$ by induction, we can perform *case analysis* on x.

Remark 2.38 (Notation). Coproducts associate to the left: the notation $A + B + C$ stands for $(A + B) + C$.

Remark 2.39. The elimination principle for coproduct types, for a constant family C, makes the diagram in Fig. 2.2 commute judgmentally.

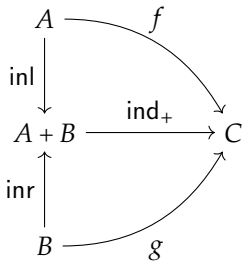


Figure 2.2: Universal property of coproduct types.

Definition 2.40. Let A, A', B and B' be types. Given functions $f : A \to A'$ and $g : B \to B'$, the function

$$f + g : A + B \to A' + B'$$

is defined by induction, so that $(f + g)(\mathsf{inl}(a)) :\equiv \mathsf{inl}(f(a))$ and $(f + g)(\mathsf{inr}(b)) :\equiv \mathsf{inr}(g(b))$ for every $a : A$ and $b : B$.

Definition 2.41. For types $A, B : \mathcal{U}$, the families $\mathsf{is\_inl}, \mathsf{is\_inr} : A + B \to \mathcal{U}$ are defined by the elimination principle for coproducts:

$$\mathsf{is\_inl}(\mathsf{inl}(a)) :\equiv \mathbf{1}, \qquad \mathsf{is\_inr}(\mathsf{inl}(a)) :\equiv \mathbf{0},$$
$$\mathsf{is\_inl}(\mathsf{inr}(b)) :\equiv \mathbf{0}, \qquad \mathsf{is\_inr}(\mathsf{inr}(b)) :\equiv \mathbf{1},$$

for every $a : A$ and $b : B$. A function $\mathsf{inl}^{-1} : \Pi\,(x : A + B)\,.\,\mathsf{is\_inl}(x) \to A$ is also defined by the same elimination principle, with $\mathsf{inl}^{-1}(\mathsf{inl}(a), l) :\equiv a$ for every $a : A$, and $\mathsf{inl}^{-1}(\mathsf{inr}(b), l)$ obtained *ex falso* for every $b : B$, since $l : \mathsf{is\_inl}(\mathsf{inr}(b)) \equiv \mathbf{0}$. Specularly, we define a function $\mathsf{inr}^{-1} : \Pi\,(x : A + B)\,.\,\mathsf{is\_inr}(x) \to B$.

Coproducts allow us to define types with a finite but arbitrary number of terms. For example, in $(\mathbf{1} + \mathbf{1}) + (\mathbf{1} + \mathbf{1})$ we find the four distinct terms $\mathsf{inl}(\mathsf{inl}(*))$, $\mathsf{inl}(\mathsf{inr}(*))$, $\mathsf{inr}(\mathsf{inl}(*))$ and $\mathsf{inr}(\mathsf{inr}(*))$. We will use the elimination rule of $\mathbb{N}$ to define canonical finite types uniformly.

Definition 2.42. The function $\mathsf{add} : \mathcal{U} \to \mathcal{U}$ is defined by $\mathsf{add}(A) :\equiv A + \mathbf{1}$.

Definition 2.43 (Canonical finite types)**.** The family $[-] : \mathbb{N} \to \mathcal{U}$ of **canonical finite types** is defined by induction, with:

$$[0] :\equiv \mathbf{0}$$
$$[n + 1] :\equiv \mathsf{add}([n]) \equiv [n] + \mathbf{1}.$$

Syntax of Inductive Types

As the elimination principle and computation rules of an inductive type T can be inferred from the list of its constructors, the type can be presented using the following, widely used syntax:

$$T ::= \text{[list of typed constructors, separated by "|"]}.$$

For example, we have:

$$\mathbf{1} ::= * : \mathbf{1}; \qquad \mathbf{0} ::= ; \qquad A + B ::= \mathsf{inl} : A \to A + B \mid \mathsf{inr} : B \to A + B;$$
$$\mathbb{N} ::= 0 : \mathbb{N} \mid \mathsf{s} : \mathbb{N} \to \mathbb{N}; \quad \mathsf{list}(X) ::= \mathsf{nil} : \mathsf{list}(X) \mid \mathsf{cons} : X \to \mathsf{list}(X) \to \mathsf{list}(X).$$

The elimination principle of an inductive type T will be denoted by

$$\mathsf{ind}_T : \Pi\,(C : T \to \mathcal{U})\,.\,(\ldots) \to \Pi\,(t : T)\,.\,C(t);$$

its non-dependent version will be denoted by $\mathsf{rec}_T : \Pi\,(C : \mathcal{U})\,.\,(\ldots) \to T \to C$.

The functions ind_T and rec_T take as arguments the target family or type C, a number of terms corresponding to the constructors of T, and a term in $t : T$. When defining a function using the elimination principle of T, we will declare it as a definition *by induction* on t, where the case analysis ("*pattern matching*") is performed by providing the terms corresponding to the constructors of T, i.e., by stating its computation rules. For example, we can define a function $f : \mathbb{N} + \mathbf{1} \to \mathbb{N}$ by induction on $x : \mathbb{N} + \mathbf{1}$ (using rec_+), declaring $f(\mathsf{inl}(n)) :\equiv n$ for every $n : \mathbb{N}$ and $f(\mathsf{inr}(y)) :\equiv 0$ for every $y : \mathbf{1}$. If the list of constructors is long, or the definitions of the arguments of ind_T or rec_T is complex, we might refer to the argument corresponding to a constructor γ as a term γ' of the correct type. If one or more constructors give the elimination principle a recursive nature (such as s for $\mathbb{N}$ or cons for $\mathsf{list}(X)$), we might present the definition recursively. For instance, a function $g : \mathsf{list}(X) \to \mathbb{N}$ returning the length of a list over a type X might be defined by stating $g(\mathsf{nil}) :\equiv 0$ and $g(x :: l) :\equiv g(l) + 1$ for every $x : X$, or alternatively by declaring $\mathsf{nil}' :\equiv 0 : \mathbb{N}$ and $\mathsf{cons}' :\equiv (x \mapsto \mathsf{s}) : X \to \mathbb{N} \to \mathbb{N}$. We will use the same syntax for higher inductive types in Section 2.6.

Families $T : I \to \mathcal{U}$ of inductive types may be *indexed* by a type I; in general, the more complex the structure of I is, the more interesting the family is. An example is the following inductive family $T : \mathbb{N} \to \mathcal{U}$, defined by the constructors:

$$T_{(-)} ::= r : T_0 \mid s : \Pi\,(n : \mathbb{N})\,.\,T_n \to T_{n+2},$$

so the types T_1, T_3, ... are *a priori* (and provably) empty. The elimination principle of $T_{(-)}$ states that, given an indexed family $C : \Pi\,(n : \mathbb{N})\,.\,T_n \to \mathcal{U}$ of types over T, a function $\mathsf{ind}_T : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(x : T_n)\,.\,C_n(x)$ can be produced by exhibiting:

- a term $r' : C_0(r)$; and
- a term $s' : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(x : T_n)\,.\,C_n(x) \to C_{n+2}(s(n, x))$;

this will compute $\mathsf{ind}_T(0, r) \equiv r'$ and $\mathsf{ind}_T(n+2, s_n(x)) \equiv s'(n, x, \mathsf{ind}_T(n, x))$ for every $n : \mathbb{N}$ and $x : T_n$. The non-dependent version of this elimination principle accordingly states that, for a family $C : \mathbb{N} \to \mathcal{U}$, a function $\Pi\,(n : \mathbb{N})\,.\,T_n \to C_n$ can be obtained once given:

- a term $r' : C_0$;
- a term $s' : \Pi\,(n : \mathbb{N})\,.\,C_n \to C_{n+2}$,

computing $\mathsf{rec}_T(0, r) \equiv r'$ and $\mathsf{rec}_T(n+2, s_n(x)) \equiv s'(n, \mathsf{rec}_T(n, x))$ for every $n : \mathbb{N}$ and $x : T_n$.

Remark 2.44. We stress that the elimination principle for inductive families (in either its dependent or non-dependent version) treats each member of the family *uniformly*, so it cannot be directly applied to produce e.g. a function $T_m \to C$ for a chosen $m : \mathbb{N}$. However, once established a notion of identity between natural numbers, we will be able to use such an elimination principle to obtain a function

$$\mathsf{rec}_T^m : \Pi\,(n : \mathbb{N})\,.\,(n = m) \to T_n \to C;$$

this can then be applied to $m : \mathbb{N}$ and a term in $m = m$ (which can always be constructed). The notion of identity we will use is described in the following section.

2.4 Identity Types

Identity types are a class of types in MLTT and HoTT. A term in an identity type embodies the notion of *equality* between two terms in a type. Although, as we will see, identity types are inductive types with only a constructor for "reflexivity" (respecting the idea that any term is always equal to itself), the structure of identity types in HoTT is rich and allows us, by iteration, to describe "higher" constructions.

Definition 2.45 (Identity types)**.** Given a type A and a term a, the family of **identity types** $\mathsf{Id}_a : A \to \mathcal{U}$, parametrized by A, is inductively defined with one constructor $\mathsf{refl}_a : \mathsf{Id}_a(a)$, which is called *identity path*, *reflexivity* or *trivial path*. The elimination principle states that, given a family $C : \Pi\,(x : A)\,.\,\mathsf{Id}_a(x) \to \mathcal{U}$ and a term

$r : C(a, \mathsf{refl}_a)$, there is a function

$$\mathsf{ind}_{\mathsf{Id},a} : \Pi\,(x : A)\,.\,\Pi\,(p : \mathsf{Id}_a(x))\,.\,C(a,p)$$

satisfying the computation rule $\mathsf{ind}_{\mathsf{Id},a}(a, \mathsf{refl}_a) \equiv r$.

Remark 2.46 (Notation). For $a, x : A$, we denote the type $\mathsf{Id}_a(x)$ with $a =_A x$ or simply $a = x$ when the type of a (and of x) is understood. For $a, x : A$, a term $p : a = x$ is called an *identification* or **path** between a and x in the type A; the latter, which we use most often in this thesis, reflects the (homotopical) interpretation of types as spaces (see Remark 2.56). Proofs using the elimination rule of identity types will be refer to as proofs "by induction on the path p". In this thesis, we will frequently use expressions such as "a path (in) $a = b$ is defined", without reserving a specific name for the term we define, in order not to overload this text with notation.

A path $p : a =_A a$ is also called a **loop** at $a : A$. The elimination principle for identity types cannot be directly applied to produce a function out of a type of loops (or a type of paths whose endpoints are both fixed), for the reason explained in Remark 2.44.

The construction of identity types can be iterated: for example, if $p, q : a =_A b$, we can form a type $p =_{(a=_A b)} q$, whose terms are dubbed 2-*paths*. In general, we will talk about *n-paths*, emphasizing their position in the tower of identity types on A (we refer to Fig. 1.3 in the introduction).

On occasion, we will display mixed chains of identities $=$ and judgmental equalities $\equiv$; for instance, if we need to construct a path $a = b$ and we have a path $p : a' = b'$ with $a' \equiv a$ and $b' \equiv b$, we will print $p : a \equiv a' = b' \equiv b$.

With this notion of identity, we are now able to prove the following "uniqueness" property of the constructors of inductive types.

Lemma 2.47. *There are paths between: every term in* $\mathbf{1}$ *and* $\star$*; every term in* $\mathbf{2}$ *and either* yes *or* no*; every term in a coproduct type and either* $\mathsf{inl}(a)$ *or* $\mathsf{inr}(b)$ *for some terms* a *and* b *of the summands; every term in a* Σ*-type and some pair; every term in* $\mathbb{N}$ *and either* 0 *or* $\mathsf{s}(n)$ *for some* $n : \mathbb{N}$*; every list in* $\mathsf{list}(X)$ *for some type* X *and either* nil *or* $x :: l$ *for some* $x : X$ *and* $l : \mathsf{list}(X)$.

Proof. All claims are proved using the elimination rules for the respective types. For example, induction on $\mathbf{2}$ for the family $C : \mathbf{2} \to \mathcal{U}$ defined as $C :\equiv (x \mapsto (x = \mathsf{yes}) + (x = \mathsf{no}))$ provides a term $\mathsf{ind}_{\mathbf{2}}(x) : (x = \mathsf{yes}) + (x = \mathsf{no})$ if we are able to find terms in $C(\mathsf{yes})$ and $C(\mathsf{no})$; these are given by $\mathsf{inl}(\mathsf{refl}_{\mathsf{yes}})$ as $\mathsf{inr}(\mathsf{refl}_{\mathsf{no}})$ respectively. □

Path Algebra

The given definition of identity types allows us to view a type A as having the structure of an ∞-groupoid: specifically, operations of inverse and composition (concatenation) of paths can be defined, satisfying relevant coherence laws. The following lemma defines such operations and enumerates a selection of statements concerning the higher groupoid structure of a type.

Definition 2.48 (Groupoid structure of types)**.** Let A be a type, and a, b, c and $d : A$.

(i) For every $p : a = b$, a path $p^{-1} : b = a$ (*inverse path*) is defined so that $\mathsf{refl}_a^{-1} \equiv \mathsf{refl}_a$;

(ii) for every $p : a = b$ and $q : b = c$, a path $p \cdot q : a = c$ (*concatenation*) is defined so that $\mathsf{refl}_a \cdot \mathsf{refl}_a \equiv \mathsf{refl}_a$;

(iii) for every $p : a = b$, a 2-path $\mathsf{inv\_inv}(p) : (p^{-1})^{-1} = p$ in A is defined (the inverse operation is an involution);

(iv) for every $p : a = b$, $q : b = c$ and $r : c = d$, a 2-path $(p \cdot q) \cdot r = p \cdot (q \cdot r)$ in A is defined (*associativity* of concatenation);

(v) for every $p : a = b$, two 2-paths $\mathsf{refl}_a \cdot p = p$ and $p \cdot \mathsf{refl}_b = p$ in A are defined (*left and right unitality* for concatenation);

(vi) for every $p : a = b$, two 2-paths $p \cdot p^{-1} = \mathsf{refl}_a$ and $p^{-1} \cdot p = \mathsf{refl}_b$ in A are defined (*inverse laws* for concatenation);

(vii) for every p_1, $p_2 : a = b$, q_1, $q_2 : b = c$ and for every 2-paths $r : p_1 = p_2$ and $s : q_1 = q_2$, a 2-path $r \cdot_{\mathsf{h}} s : p_1 \cdot q_1 = p_2 \cdot q_2$ in A is defined (*horizontal composition*, where vertical composition is concatenation of 2-paths); in case r (resp. s) is the identity 2-path, this is called *left* (resp. *right*) *whiskering*;

(viii) for every $l : a = b$ and $p, q : b = c$, a function $(l \cdot p = l \cdot q) \to (p = q)$ (*left cancelling*) is defined; similarly, cancelling can be performed on the right;

(ix) horizontal composition is associative and satisfies the unit laws; together with concatenation, it satisfies the interchange laws, i.e. a 3-path

$$(r_1 \cdot_{\mathsf{h}} s_1) \cdot (r_2 \cdot_{\mathsf{h}} s_2) = (r_1 \cdot r_2) \cdot_{\mathsf{h}} (s_1 \cdot s_2) \tag{2.49}$$

in A is constructed for composable 2-paths r_i, s_i (Fig. 2.3), and 1-coherence diagrams (associativity pentagon, associativity-unitality triangle with respect to (iv) and (v)).

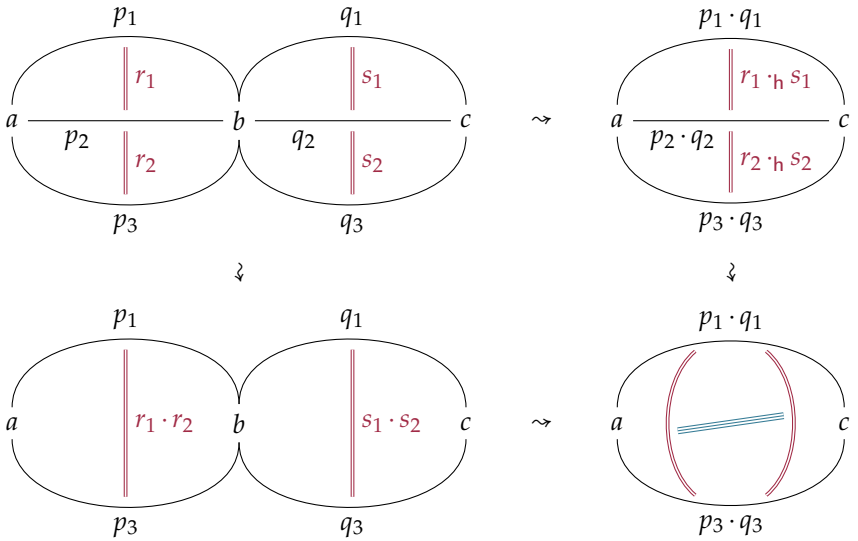


Figure 2.3: The interchange law for 2-paths provides a 3-path (in blue) between the two 2-paths in the bottom-right diagram (in red), which are $(r_1 \cdot_h s_1) \cdot (r_2 \cdot_h s_2)$ and $(r_1 \cdot r_2) \cdot_h (s_1 \cdot s_2)$.

All these terms are obtained by induction on the given paths or higher paths, i.e. via the elimination rule for identity types. For example, in (i), the family

$$C : \Pi\,(x : A)\,.\,\mathsf{Id}_a(x) \to \mathcal{U}$$

is defined for every a to be $C :\equiv (x \mapsto (p \mapsto x =_A a))$, and the term $r : C(a, \mathsf{refl}_a) \equiv (a = a)$ is defined to be $r :\equiv \mathsf{refl}_a$, so the inverse is defined for *every* path and is such that $\mathsf{refl}_a^{-1} :\equiv \mathsf{refl}_a$.

Remark 2.50. It would be tempting to complete the list in Definition 2.48 with a general (and deliberately unspecific) item:

(x) "all" higher coherence relations are satisfied;

sadly, no type is known to express this statement. It is believed that the axiomatisation of so-called *semi-simplicial types* would allow the theory to express this and other higher coherence statements and, in particular, to describe ∞-groupoids and $(\infty, 1)$-categories internally to the theory [Buc19; RS17; KA18]. Even though this problem is at the present time still open, for the purposes of this thesis (which will never trespass groupoid coherence at the level of 2-paths) we can still work under the informal assumption that path induction proves all the coherence statements we need about the higher path structure of a type.

Remark 2.51 (Notation). In this exposition we will not need to emphasize the distinction between differently-associated concatenations of the same paths, and we will use the unbracketed notation $p \cdot q \cdot \ldots \cdot r$ for the concatenation of three or more paths.

Functions between types act functorially with respect to paths, in the following sense.

Definition 2.52 (Application of functions and transport)**.** Let A be a type. For every type B, function $f : A \to B$ and terms $x, y : A$, we construct a function

$$[f] : (x = y) \to (f(x) = f(y)) \tag{2.53}$$

by path induction, declaring $[f](\mathsf{refl}_x) :\equiv \mathsf{refl}_{f(x)}$, i.e., $[f]$ preserves identity paths judgmentally. For every family $C : A \to \mathcal{U}$ and terms $x, y : A$, we use path induction again to construct a function

$$(-)^C_* : (x = y) \to (C(x) \to C(y)), \tag{2.54}$$

named **transport**, such that $(\mathsf{refl}_x)^C_* \equiv \mathsf{id}_{C(x)}$ for all $x : A$. Finally, for every function $f : \Pi\,(x : A)\,.\,C(x)$ and terms $x, y : A$, we construct a function

$$[f]^{\mathsf{d}} : \Pi\,(p : x = y)\,.\,\left(p^C_*(f(x)) =_{C(y)} f(y)\right) \tag{2.55}$$

by path induction, defining $[f]^{\mathsf{d}}(\mathsf{refl}_x) :\equiv \mathsf{refl}_{f(x)}$ and hence making $[f]^{\mathsf{d}}$ preserve identity paths judgmentally. The functions $[f]$ in (2.53) and $[f]^{\mathsf{d}}$ in (2.55) are the **application to a path** of a (dependent) function.[2]

Remark 2.56. The function $(-)^C_*$ in (2.54) represents transport of a term in a fiber along a path, in the following sense. By path induction, we can prove that, for every $a_1, a_2 : A$ and $p : a_1 = a_2$, there is a function

$$\mathsf{lift}(p) : \Pi\,(z : B(a_1))\,.\,\langle a_1, z\rangle =_{(\Sigma(x:A).\,B(x))} \langle a_2, p^B_*(z)\rangle \tag{2.57}$$

(Fig. 2.4); for this reason, one can see the type $\Sigma\,(x : A)\,.\,B(x)$ as the total space of a *fibration* (given by the projection to A), where the base is given by the type A, the fiber of any $x : A$ is the type $B(x)$, and the function in (2.57) yields a lifting to the total space of any path in A. A dependent function $f : \Pi\,(x : A)\,.\,B(x)$ is then a section of the fibration. We will see in Lemma 2.107 that the fibers of the endpoints of a path in A are equivalent, for a suitable notion of equivalence (Section 2.5). This interpretation justifies the following definition.

[2] In literature, the functions $[f]$ and $[f]^{\mathsf{d}}$ are often denoted by ap_f and apd_f, respectively.

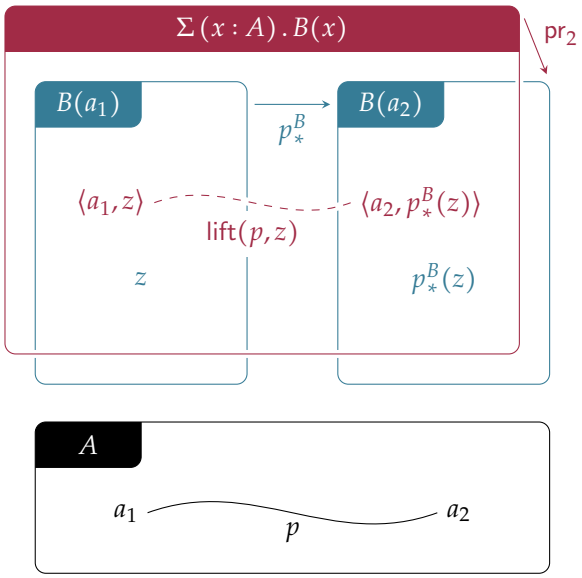


Figure 2.4: Lifting of a path to the total space of a fibration.

Definition 2.58. Let $f : A \to B$ be a function of types and $b : B$. The (homotopy-)**fiber** of f over b is the type

$$\mathsf{fib}_f(b) :\equiv \Sigma\,(a : A)\,.\,(f(a) =_B b).$$

In particular, if $A : \mathcal{U}$ and $B : A \to \mathcal{U}$, the fiber of $\mathsf{pr}_1 : (\Sigma\,(x : A)\,.\,B(x)) \to A$ over any $a : A$ is the type

$$\Sigma\,(y : \Sigma\,(x : A)\,.\,B(x))\,.\,(\mathsf{pr}_1(y) =_A a),$$

which can be shown to be equivalent to $B(a)$, for a notion of equivalence described in Section 2.5.

Remark 2.59 (Pathovers). A path in the fiber of a family $C : A \to \mathcal{U}$ that belongs to a type of the form $p^C_*(u) = v$ for some path p in A is also called a path *over* p, or *pathover* [LB15].

Application of functions and dependent functions to paths satisfies certain properties – for example, it respects concatenation of paths and composition of functions, albeit not judgmentally. This fits in the categorical interpretation (as ∞-groupoids) of types: functions between types are actually *functors*. We will heavily rely on this interpretation in order to reach the main results of this thesis. We summarize some key properties in the following lemma.

Definition 2.60 (Some path algebra)**.** For types $A, B, C : \mathcal{U}$ and functions $f : A \to B$ and $g : B \to C$:

(i) given a path p in A, we define a 2-path $[\mathsf{id}_A](p) = p$;

(ii) given composable paths p and q in A, a 2-path $[f](p \cdot q) = [f](p) \cdot [f](q)$ is defined;

(iii) given a path p in A, we define a 2-path $[f](p^{-1}) = [f](p)^{-1}$;

(iv) given a path p in A, we define a 2-path $[g \circ f](p) = [g]([f](p))$;

(v) if $f \equiv (z \mapsto b)$ for some $b : B$, i.e., if f is constant, we define a 2-path $[f](p) = \mathsf{refl}_b$ for every path p in A;

Given a type $A : \mathcal{U}$ and a family $C : A \to \mathcal{U}$:

(vi) for paths $p : x =_A y$ and $q : y =_A z$, we derive a path between functions $(p \cdot q)^C_* =_{C(x) \to C(z)} q^C_* \circ p^C_*$;

(vii) given paths $p : x =_A y$ and $q : y =_A z$, terms $x' : C(x)$, $y' : C(y)$ and $z' : C(z)$ and pathovers $p' : p^C_*(x') = y'$ and $q' : q^C_*(y') = z'$, we define a pathover

$$p' \cdot^{\mathsf{d}} q' : (p \cdot q)^C_*(x') = z'$$

so that, if p and q are identity paths at x, then $p' \cdot^{\mathsf{d}} q' \equiv p' \cdot q'$ (see Fig. 2.5);

(viii) given a path $p : x =_A y$, terms $x' : C(x)$ and $y' : C(y)$ and a pathover $q : p^C_*(x') = y'$, we define a pathover $(p^{-1})^C_*(y') = x'$;

(ix) if the family C is constant at $B : \mathcal{U}$, for a path $p : x =_A y$ and a term $b : B$, we define a pathover $\mathsf{tr\_const}(p, b) : p^C_*(b) = b$, respecting identity paths judgmentally.

Given types $A, A' : \mathcal{U}$, families $C : A \to \mathcal{U}$ and $C' : A' \to \mathcal{U}$, a path $p : x =_A y$, terms $x' : C(x)$ and $y' : C(y)$ and a pathover $q : p^C_*(x') = y'$,

(x) for any function $f : A \to A'$ and $g : \Pi(a : A).\, C(a) \to C'(f(a))$, we define a pathover (depicted in Fig. 2.6)

$${}^{f|g}q : ([f](p))^{C'}_*(g_x(x')) =_{C'(f(y))} g_y(y')$$

such that, if p is the identity path on x, there is a 2-path ${}^{f|g}q = [g_x](q)$.

Given types $A, B : \mathcal{U}$ and functions $f, g : A \to B$, we can consider the family $C : A \to \mathcal{U}$, $C(z) :\equiv f(z) = g(z)$ of identity types; then:

(xi) given paths $p : x =_A y$, $q_x : C(x)$ and $q_y : C(y)$, we define a function

$$\left(q_x \cdot [g](p) =_{(f(x)=g(y))} [f](p) \cdot q_y\right) \to \left(p_*^C(q_x) = q_y\right). \tag{2.61}$$

For a type $A : \mathcal{U}$, a family $C : A \to \mathcal{U}$ and a function $f : \Pi\,(x : A)\,.\,C(x)$:

(xii) for composable paths p and q in A, a 2-path $[f]^{\mathsf{d}}(p \cdot q) = [f]^{\mathsf{d}}(p) \cdot^{\mathsf{d}} [f]^{\mathsf{d}}(q)$ is defined;

(xiii) if the family C is constant at $B : \mathcal{U}$, given a path $p : x =_A y$, we define a 2-path $[f]^{\mathsf{d}}(p) = \mathsf{tr\_const}(p, f(x)) \cdot [f](p)$.

All terms are constructed by path induction. We note that the pathover $p' \cdot^{\mathsf{d}} q'$ in (vii) can be, alternatively, defined directly by:

$$\begin{aligned}
(p \cdot q)_*^C(x') &= q_*^C(p_*^C(x')) && \text{by (vi)} \\
&= q_*^C(y') && \text{by } [q_*^C](p') \\
&= z' && \text{by } q'.
\end{aligned}$$

We also remark that the function in (2.61) will be an equivalence, in the sense of Definition 2.89, as shown in Lemma 2.102.

We will frequently use application of functions in two variables, as expressed in the following lemma, which is also proved by path induction.

Lemma 2.62. *Given types A, B and $C : \mathcal{U}$, a function $f : A \to B \to C$ and terms $a_1, a_2 : A$ and $b_1, b_2 : B$, a function*

$$[f(-,-)] : (a_1 = a_2) \to (b_1 = b_2) \to (f(a_1, b_1) = f(a_2, b_2)), \tag{2.63}$$

is defined, computing to the identity path on identity paths in A and B, such that

$$[f(-,b)](p) = [x \mapsto f(x,b)](p) \qquad \text{and} \qquad [f(a,-)](q) = [x \mapsto f(a,x)](q)$$

for $a : A$, $b : B$, $p : x_1 =_A y_1$ and $q : x_2 =_B y_2$, and satisfying the interchange law:

$$[f(-,-)](p_1, q_1) \cdot [f(-,-)](p_2, q_2) = [f(-,-)](p_1 \cdot p_2, q_1 \cdot q_2) \tag{2.64}$$

for paths p_1, p_2, q_1 and q_2 making the expression well-typed.

Remark 2.65 (Notation). If a function $\boxtimes : A \to A \to A$ is to represent a product, we will generally use the infix notation $a_1 \boxtimes a_2 :\equiv \boxtimes(a_1, a_2)$ for $a_1, a_2 : A$. Similarly, the notation $p_1 \boxtimes p_2 :\equiv [- \boxtimes -](p_1, p_2)$ will be used for paths p_1, p_2 in A.

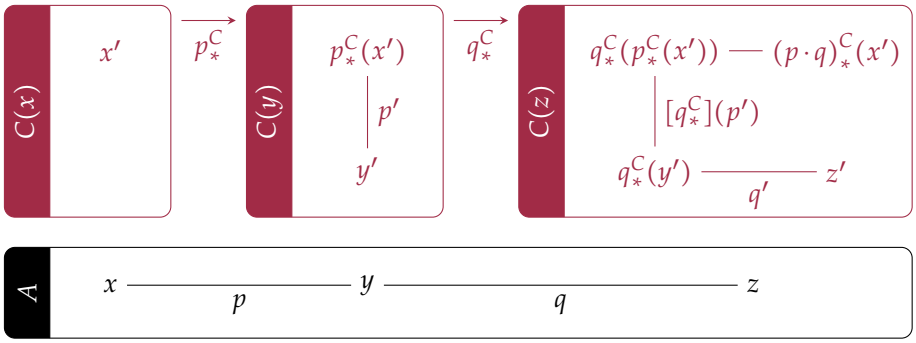


Figure 2.5: Construction of the pathover $p' \cdot^{\mathsf{d}} q' : (p \cdot q)^C_*(x') = z'$ in Definition 2.60(vii). The unlabelled horizontal path in $C(z)$ is given by Definition 2.60(vi).

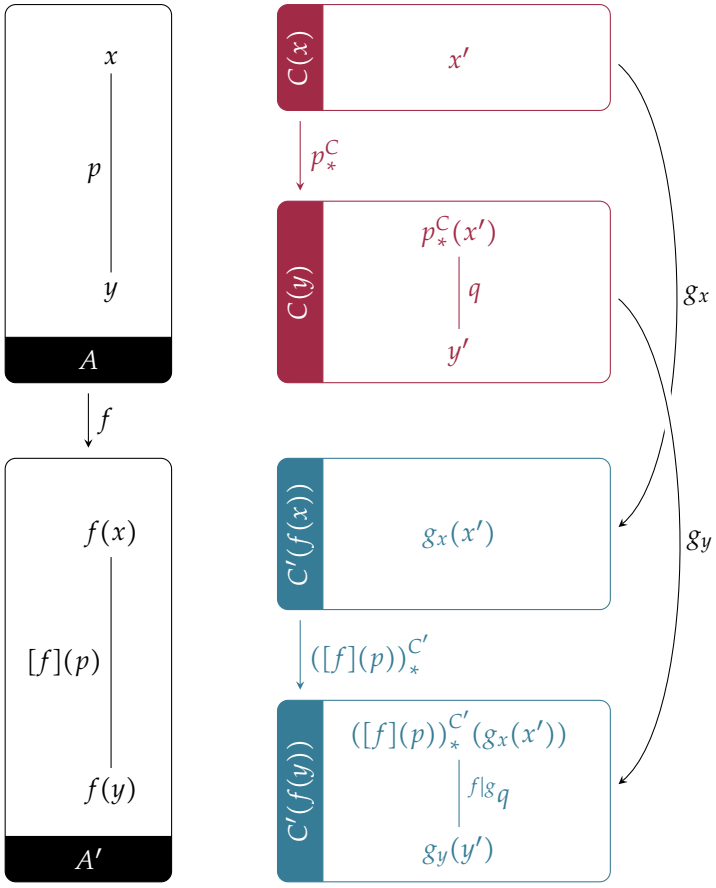


Figure 2.6: Depiction of ${}^{f|g}q$ in Definition 2.60(x). By induction on p, it is enough to provide a path $g_x(x') =_{C'(f(x))} g_x(y')$ for every term $x : A$, terms $x', y' : C(x)$ and path $q : x' = y'$; this is given by $[g_x](q)$, which, itself, is defined by induction on q.

It is not hard to imagine countless other properties that transport of paths and application of functions satisfy, all proved by path induction; we will refer to those and to the ones stated in Definition 2.60 as "path algebra".

Paths in Σ-types deserve particular attention. In the following lemmata, we assume $A : \mathcal{U}$ and $P : A \to \mathcal{U}$.

Lemma 2.66. *Let $x, x' : A$, $y : P(x)$ and $y' : P(x')$. A path $p : x = x'$ and a pathover $q : p^P_*(y) = y'$ (i.e. a term in $\Sigma\,(p : x = x')\,.\,p^P_*(y) = y'$) are enough to define a path*

$$\langle p, q\rangle : \langle x, y\rangle =_{(\Sigma(a:A).\,P(a))} \langle x', y'\rangle.$$

Conversely, given $z, z' : \Sigma\,(a : A)\,.\,P(a)$, a path $r : z = z'$ determines the paths

$$\overline{\mathsf{pr}_1}(r) : \mathsf{pr}_1(z) = \mathsf{pr}_1(z'), \qquad \overline{\mathsf{pr}_2}(r) : (\overline{\mathsf{pr}_1}(r))^P_*(\mathsf{pr}_2(z)) = \mathsf{pr}_2(z').$$

The definitions of $\langle -, -\rangle$, pr_1 and pr_2 are such that there are 2-paths

$$\langle \overline{\mathsf{pr}_1}(r), \overline{\mathsf{pr}_2}(r)\rangle = r, \qquad \overline{\mathsf{pr}_1}\langle p, q\rangle = p, \qquad \overline{\mathsf{pr}_2}\langle p, q\rangle = q. \tag{2.67}$$

Proof. For the first claim: by induction on p, we need to check that, for a pathover $q : (\mathsf{refl}_x)^P_*(y) \equiv y =_{P(x)} y'$ there is a path $\langle x, y\rangle = \langle x, y'\rangle$ in the Σ-type. By induction on q it is enough to find a path $\langle x, y\rangle = \langle x, y\rangle$, which can be given by $\mathsf{refl}_{\langle x,y\rangle}$. The computation rule of path induction then states that $\langle \mathsf{refl}_x, \mathsf{refl}_y\rangle \equiv \mathsf{refl}_{\langle x,y\rangle}$. As for the second claim, by induction on r we define $\overline{\mathsf{pr}_1}(\mathsf{refl}_z) :\equiv \mathsf{refl}_{\mathsf{pr}_1(z)}$ and $\overline{\mathsf{pr}_2}(\mathsf{refl}_z) :\equiv \mathsf{refl}_{\mathsf{pr}_2(z)}$. The 2-paths in (2.67) are then also found by induction on p, q and r. □

Remark 2.68. We emphasize that the definition of $\overline{\mathsf{pr}_1}$ in Lemma 2.66 matches the one of $[\mathsf{pr}_1]$. Hence, from (2.67), we get that $[\mathsf{pr}_1]\langle p, q\rangle = p$. For $z, z' : \Sigma\,(a : A)\,.\,P(a)$, Lemma 2.66 establishes an equivalence between the identity type $z = z'$ and the type $\Sigma\,(p : \mathsf{pr}_1(z) = \mathsf{pr}_1(z'))\,.\,p^P_*(\mathsf{pr}_2(z)) = \mathsf{pr}_2(z')$, in the sense of Section 2.5.

Lemma 2.69. *The following holds:*

(i) for $p_1 : x_1 =_A x_2$, $p_2 : x_2 =_A x_3$, $q_1 : (p_1)^P_*(y_1) = y_2$ and $q_2 : (p_2)^P_*(y_2) = y_3$ (with $x_i : A$ and $y_i : P(x_i)$), we construct a 2-path

$$\langle p_1, q_1\rangle \cdot \langle p_2, q_2\rangle = \langle p_1 \cdot p_2,\ q_1 \cdot^{\mathsf{d}} q_2\rangle;$$

(ii) given $A' : \mathcal{U}$ and $P' : A' \to \mathcal{U}$, a path $\langle p, q\rangle : \langle x, y\rangle =_{(\Sigma(a:A).\,P(a))} \langle x', y'\rangle$ as in Lemma 2.66 and a function $\langle f, g\rangle : \Sigma\,(a : A)\,.\,P(a) \to \Sigma\,(a' : A')\,.\,P'(a')$ as in Definition 2.18, we construct a 2-path

$$[\langle f, g\rangle]\langle p, q\rangle = \langle [f](p),\ {}^{f|g}q\rangle$$

using the notation of Definition 2.60(x);

(iii) given paths $\langle p, q\rangle, \langle p', q'\rangle : \langle x, y\rangle =_{(\Sigma(a:A).\, P(a))} \langle x', y'\rangle$*, and 2-paths*

$$r : p =_{(x=x')} p' \qquad\qquad s : q =_{(p_*^P(y)=y')} [(-)_*^P(y)](r)\cdot q',$$

we construct a 2-path $\langle p, q\rangle = \langle p', q'\rangle$.

Proof. All claims are proved by path induction. □

Finally, it is useful to notice that identity types and transport can help us to discriminate between different constructors of an inductive type.

Lemma 2.70. *There is no path between:* yes *and* no *in* $\mathbf{2}$*;* $\mathsf{inl}(a)$ *and* $\mathsf{inr}(b)$ *in a coproduct* $A + B$ *of types, for any* $a : A$ *and* $b : B$*;* 0 *and* $\mathsf{s}(n)$ *in* $\mathbb{N}$ *for any* $n : \mathbb{N}$*;* nil *and* $x :: l$ *in* $\mathsf{list}(X)$ *for any* $x : X$ *and* $l : \mathsf{list}(X)$.

Proof. Once again, all statements are proved in the same way. For example, in order to show that there is no path betwen yes and no in $\mathbf{2}$, we need to produce a function $(\mathsf{yes} = \mathsf{no}) \to \mathbf{0}$. The function $\mathsf{rec}_{\mathbf{2}} : \Pi\,(A : \mathcal{U})\,.\, A \to A \to \mathbf{2} \to A$ allows us to define a family of types $C :\equiv \mathsf{rec}_{\mathbf{2}}(\mathcal{U}, \mathbf{1}, \mathbf{0}) : \mathbf{2} \to \mathcal{U}$ such that $C(\mathsf{yes}) \equiv \mathbf{1}$ and $C(\mathsf{no}) \equiv \mathbf{0}$. Given a path $p : \mathsf{yes} = \mathsf{no}$, we have $p_*^C : C(\mathsf{yes}) \to C(\mathsf{no})$, hence $p_*^C(*) : \mathbf{0}$. □

Lemma 2.71. *Referring to Definition 2.41, for every* $A, B : \mathcal{U}$*, we construct dependent functions of types*

$$\Pi\,(x : A + B)\,.\ \Pi\,(l : \mathsf{is\_inl}(x))\,.\, \mathsf{inl}(\mathsf{inl}^{-1}(x, l)) = x \quad \textit{and}$$
$$\Pi\,(x : A + B)\,.\, \Pi\,(r : \mathsf{is\_inr}(x))\,.\, \mathsf{inr}(\mathsf{inr}^{-1}(x, r)) = x.$$

Proof. The first function is defined by induction on x, with $\mathsf{inl}(\mathsf{inl}^{-1}(\mathsf{inl}(a), l)) \equiv \mathsf{inl}(a)$ for every $a : A$, and $\mathsf{inl}(\mathsf{inl}^{-1}(\mathsf{inr}(b), l)) = \mathsf{inr}(b)$ obtained *ex falso* for every $b : B$, as $l : \mathsf{is\_inl}(\mathsf{inr}(b)) \equiv \mathbf{0}$. The second function is defined analogously. □

Lemma 2.72. *Let* $A, B : \mathcal{U}$*. We construct the dependent functions:*

$$f_A : \Pi\,(a_1, a_2 : A)\,.\, (\mathsf{inl}(a_1) =_{(A+B)} \mathsf{inl}(a_2)) \to (a_1 =_A a_2) \quad \textit{and}$$
$$f_B : \Pi\,(b_1, b_2 : B)\,.\, (\mathsf{inr}(b_1) =_{(A+B)} \mathsf{inr}(b_2)) \to (b_1 =_B b_2).$$

Proof. What follows is part of a standard "encode-decode" proof, which we will better describe in Section 3.4. We define a family $C : (A + B) \to (A + B) \to \mathcal{U}$ using the elimination principle of the coproduct:

$$\begin{aligned}
C(\mathsf{inl}(a_1), \mathsf{inl}(a_2)) &:\equiv (a_1 =_A a_2) && \text{for every } a_1, a_2 : A\\
C(\mathsf{inl}(a), \mathsf{inr}(b)) &:\equiv \mathbf{0} && \text{for every } a : A, b : B\\
C(\mathsf{inr}(b), \mathsf{inl}(a)) &:\equiv \mathbf{0} && \text{for every } a : A, b : B\\
C(\mathsf{inr}(b_1), \mathsf{inr}(b_2)) &:\equiv (b_1 =_B b_2) && \text{for every } b_1, b_2 : B.
\end{aligned}$$

Then, a function $f : \Pi(x_1, x_2 : A + B).(x_1 = x_2) \to C(x_1, x_2)$ is produced by path induction and subsequent elimination principle of the coproduct:

$$f(\mathsf{inl}(a), \mathsf{inl}(a), \mathsf{refl}_{\mathsf{inl}(a)}) :\equiv \mathsf{refl}_a : C(\mathsf{inl}(a), \mathsf{inl}(a))$$
$$f(\mathsf{inr}(b), \mathsf{inr}(b), \mathsf{refl}_{\mathsf{inr}(b)}) :\equiv \mathsf{refl}_b : C(\mathsf{inr}(b), \mathsf{inr}(b)).$$

We can then define $f_A(a_1, a_2, p) :\equiv f(\mathsf{inl}(a_1), \mathsf{inl}(a_2), p) \equiv (a_1 =_A a_2)$, and similarly $f_B(b_1, b_2, p) :\equiv f(\mathsf{inr}(b_1), \mathsf{inr}(b_2), p) \equiv (b_1 =_B b_2)$. □

Homotopy n-Types

If the $(n+2)$-path structure of a type X is trivial, i.e., if the type X supports *all* the $(n+2)$-paths it could have, then the type X is called a (homotopy) n-type, as presented in the following definition.

Definition 2.73 (n-types)**.** A type X is called a (-1)**-type** or **proposition** if there is a path between every two terms of X, i.e. if there is a term

$$t : \mathsf{isHProp}(X) :\equiv \Pi(x, y : X).\, x = y.$$

A family of (-1)-types, i.e., a family $P : X \to \mathcal{U}$ together with a term in

$$\Pi(x : X).\ \mathsf{isHProp}(P(x)),$$

is also called a **property** about terms in X; if a term $t : P(a)$ is provided for some $a : A$, then a is said to satisfy the property P. If $P : \mathcal{U} \to \mathcal{U}$ is a property, the Σ-type $\Sigma(A : \mathcal{U}).P(a)$ is called a **subuniverse** of $\mathcal{U}$.

A type X is called an $(n+1)$**-type**, for $n \geq -1$, if $x =_X y$ is an n-type for every x, $y : X$. Notably, X is a 0**-type** or **set** if there is a 2-path between every two paths in X, i.e. if there is a term

$$t : \mathsf{isHSet}(X) :\equiv \Pi(x, y : X).\, \Pi(p, q : x = y).\, p = q;$$

families of 0-types are understood to be the interpretation of covering spaces of the indexing type [FH18]. The type X is a 1**-type** or **groupoid** if there is a term

$$t : \mathsf{isHGpd}(X) :\equiv \Pi(x, y : X).\, \Pi(p, q : x = y).\, \Pi(r, s : p = q).\, r = s,$$

i.e. if there is a 3-path between every two 2-paths in X.

A type X is called a (-2)**-type** or **contractible** if it is a nonempty proposition, i.e. if there is a term

$$t : \mathsf{isContr}(X) :\equiv X \times \mathsf{isHProp}(X);$$

assuming function extensionality (Definition 2.114), the type of $\mathsf{IsContr}(X)$ is equivalent to the type

$$\Sigma\,(x_0 : X)\,.\,\Pi\,(y : X)\,.\,x_0 = y \tag{2.74}$$

in the sense of Definition 2.89 [Uni13, Lemma 3.11.3], which we will use every time we will need to prove that a type is contractible. If $\langle c, h\rangle$ is a term of the Σ-type in (2.74), the terms $c : X$ and $h : \Pi\,(y : X)\,.\,c = y$ are called, respectively, *center* and *proof of the contraction*.

If X is an n-type, n is called the **truncation level** of X. The type expressing the n-truncatedness of X is denoted by $\mathsf{IsTrunc}_n(X)$.

Remark 2.75. Using their elimination rules, one can see that the types **0** and **1** are propositions, the latter being also contractible. The type $\mathbb{N}$ is a set; Σ-types of families of n-types over an n-type are n-types themselves; Π-types of families of n-types, via function extensionality (Definition 2.114 and Lemma 2.126), are also; coproducts of $(n+2)$-types are $(n+2)$-types, and hence all canonical finite types are 0-types [Uni13, Section 3.1].

We will liberally make use of the following lemmata throughout this thesis.

Lemma 2.76. *If X is an n-type, then X is also an $(n+1)$-type.*

Proof. Proved in [Uni13, Theorem 7.1.7]. □

Lemma 2.77. *For any type X, the type $\mathsf{IsTrunc}_n(X)$ is a (-1)-type.*

Proof. This is shown in [Uni13, Theorem 7.1.10]. □

Remark 2.78 (Paths in Σ-types with contractible fibers). If $P : A \to \mathcal{U}$ is a *property* about terms in a type A in the sense of Definition 2.73, by Lemma 2.66 we can conclude that a path $p : x =_A x'$ is enough to define a path

$$\langle x, y\rangle =_{(\Sigma(a:A).\,P(a))} \langle x', y'\rangle$$

for any $y : P(x)$, $y' : P(x')$, as a pathover is provided by the truncation level of the fibers of the Σ-type. We will often denote such a path as

$$\langle p, \ldots\rangle : \langle x, y\rangle = \langle x', y'\rangle,$$

omitting the pathover. Similarly, if all the types in the family P are 0-types, the requirement s in Lemma 2.69(iii) is automatically fulfilled.

Lemma 2.79. *Let A be a type and $P : A \to \mathcal{U}$ a family of types. Assume that $P(a)$ is an $(n+1)$-type for every $a : A$, and that for every $x, y : \Sigma(a:A).P(a)$, the type $\mathsf{pr}_1(x) =_A \mathsf{pr}_1(y)$ is an n-type. Then $\Sigma(a:A).P(a)$ is an $(n+1)$-type.*

Proof. We need to show that, for every $x, y : \Sigma(a:A).P(a)$, the type $x = y$ is an n-type. From the assumptions, using Remark 2.75, we obtain that the type

$$B :\equiv \Sigma(p : \mathsf{pr}_1(x) = \mathsf{pr}_2(x)).\, p_*^P(\mathsf{pr}_2(x)) = \mathsf{pr}_2(y)$$

is an n-type, since the type in the base is an n-type and the fibers are types of identities in an $(n+1)$-type. The type B is equivalent to the type $x = y$ (Remark 2.68); hence, the latter is an n-type. □

Given any type X, one constructs a type $\|X\|_n$ by adding all $(n+2)$-paths; this is a higher inductive type, which we will present in Definition 2.135.

2.5 Equivalences and Paths in the Universe

As shown in Section 2.4, two terms in the same type satisfy the same statements whenever a path between them can be provided; that is, given a type $A : \mathcal{U}$, a family $B : A \to \mathcal{U}$ and a path $p : x =_A y$, the operation of transport returns a term $p_*^B(u) : B(y)$ once given a term $u : B(x)$. The functions $p_*^B : B(x) \to B(y)$ and $(p^{-1})_*^B : B(y) \to B(x)$ can be seen as *inverse* to each other, in the (weak) sense that, by path algebra, we can provide a path

$$(p^{-1})_*^B(p_*^B(u)) = (p \cdot p^{-1})_*^B(u) = (\mathsf{refl}_x)_*^B(u) \equiv u$$

in $B(x)$ (and vice versa, given a term $v : B(y)$). This kind of correspondence asks for a dedicated notion of *equivalence* between $B(x)$ and $B(y)$, which we will introduce in this section.

Equivalences

Definition 2.80 (Homotopy)**.** Given a type A, a family $B : A \to \mathcal{U}$ and dependent functions $f, g : \Pi(a:A).B(a)$, a **homotopy** from f to g is a pointwise equality between the two functions, i.e. a function

$$h : \Pi(a:A).\, f(a) = g(a). \tag{2.81}$$

We will denote the type of h as $f \sim g$. The same definition applies to functions $f, g : A \to B$, if B is a type.

Definition 2.82 (Involution)**.** A function $f : A \to A$ is an **involution** if there is a homotopy $h : (f \circ f) \sim \mathsf{id}_A$.

Definition 2.83. For every $A, B : \mathcal{U}$ and $f : A \to B$, there is an identity homotopy $\mathsf{id}_f^* : f \sim f$, defined by $\mathsf{id}_f^*(a) :\equiv \mathsf{refl}_{f(a)}$ for every $a : A$. Given a homotopy $h : f \sim f'$, there is an inverse homotopy $h^{-1*} : f' \sim f$, defined by $h^{-1*}(a) :\equiv (h(a))^{-1}$ for every $a : A$. Given also a homotopy $h' : f' \sim f''$, there is a composite homotopy $h \cdot^* h'$, defined by $(h \cdot^* h')(a) :\equiv h(a) \cdot h'(a)$ for every $a : A$.

Definition 2.84. Let $A : \mathcal{U}$, $B : A \to \mathcal{U}$ and $f, g : \Pi\,(a : A)\,.\,B(a)$. There is a function

$$[-]^* : (f = g) \to (f \sim g),$$

defined, by path induction, by $[\mathsf{refl}_f]^* :\equiv \mathsf{id}_f^*$. Notice that, by definition of the application of functions, we have $[h]^*(a) \equiv [(-)(a)](h)$ for every $h : f = g$ and $a : A$.

Lemma 2.85. *The function $[-]^*$ respects composition and inverses, i.e. there are paths $[p \cdot q]^* = [p]^* \cdot^* [q]^*$ and $[p^{-1}]^* = ([p]^*)^{-1*}$.*

Proof. By induction on p and q. □

Lemma 2.86. *For $A : \mathcal{U}$, $B : A \to \mathcal{U}$ and $f, g : A \to B$, if $B(a)$ is a (-1)-type for every $a : A$, then there is a homotopy $f \sim g$.*

Proof. Immediate from the definition of (-1)-type. □

Lemma 2.87. *The function $[-]^*$ respects postcompositions, in the following sense: for types $A, B, C : \mathcal{U}$, functions $f, f' : A \to B$ and $g : B \to C$, a path $p : f = f'$ and a term $a : A$, a 2-path*

$$[[(z : A \to B) \mapsto (x : A) \mapsto g(z(x))](p)]^*(a) =_{(g(f(a)) = g(f'(a)))} [g]([p]^*(a))$$

is defined.

Proof. By induction on the path p, we only need to provide a path $\mathsf{refl}_{g(f(a))} = \mathsf{refl}_{g(f(a))}$, which can be given by $\mathsf{refl}_{\mathsf{refl}_{g(f(a))}}$. □

Definition 2.88. Let $f : A \to B$ be a function of types. A function $g : B \to A$, together with a proof that $f \circ g \sim \mathsf{id}_B$ is called a **section** of f; the function f is then said to be a **retraction** of g, and the type B is called a **retract** of A.

Definition 2.89 (Equivalence)**.** Let A and B be types. The type of **equivalences** between A and B is the type

$$A \simeq B :\equiv \Sigma\,(f : A \to B)\,.\,\Sigma\,(g : B \to A)\,.\,\Sigma\,(h_1 : f \circ g \sim \mathsf{id}_B)\,.\,\Sigma\,(h_2 : g \circ f \sim \mathsf{id}_A)\,.$$
$$\Pi\,(a : A)\,.\,h_1(f(a)) = [f](h_2(a)); \tag{2.90}$$

i.e., the types A and B are *equivalent* if there are functions $f : A \to B$ and $g : B \to A$ which are sections of each other in a "compatible" way. The components f and g in $\langle f, g, h_1, h_2, \delta\rangle : A \simeq B$ are said to be *half-adjoint* in the equivalence they define, and δ is called the *adjunction* between the homotopies h_1 and h_2. An equivalence $A \simeq A$ will also be called a **symmetry**.

Remark 2.91 (Notation). There is an obvious function $(-)^{-1} : (A \simeq B) \to (B \simeq A)$, defined by $\langle f, g, h_1, h_2, \delta\rangle^{-1} :\equiv \langle g, f, h_2, h_1, \delta'\rangle$, where the construction of the adjunction δ' is obtained by path algebra and made explicit in [Uni13, Lemma 4.2.2]. Given an equivalence $e : A \simeq B$ of types, we will be usually interested in the underlying half-adjoint functions $\mathsf{pr}_1(e) : A \to B$ and $\mathsf{pr}_1(\mathsf{pr}_2(e)) \equiv \mathsf{pr}_1(e^{-1}) : B \to A$, which we will denote, respectively, by e and e^{-1}; the context will make clear whether we will refer to an equivalence or to the underlying function. Conversely, we will say that a function $f : A \to B$ "is an equivalence" if its half-adjoint and the corresponding homotopies can be provided, i.e. if the type

$$\mathsf{IsEquiv}(f) :\equiv \Sigma\,(g : B \to A)\,.\,\Sigma\,(h_1 : \ldots, h_2 : \ldots)\,.\,\Pi\,(a : A)\,.\,h_1(f(a)) = [f](h_2(a))$$

is inhabited (so $(A \simeq B) \equiv \Sigma\,(f : A \to B)\,.\,\mathsf{IsEquiv}(f)$). For example, assuming function extensionality (Definition 2.114), the function $(-)^{-1}$ between equivalence types is, itself, an equivalence. Similarly, we will say that two types A and $B : \mathcal{U}$ "are equivalent" if an equivalence $e : A \simeq B$ can be given.

Remark 2.92. In this thesis, every time we will prove that two functions are half-adjoint in an equivalence, we will omit the proof of the adjunction between the homotopies, as a consequence of the following lemma.

Lemma 2.93. *Let $f : A \to B$; $g : B \to A$, $h_1 : f \circ g \sim \mathsf{id}_B$ and $h_2 : g \circ f \sim \mathsf{id}_A$ be given, for $A, B : \mathcal{U}$. Terms h_2' and δ are constructed, such that $\langle f, g, h_1, h_2', \delta\rangle : A \simeq B$.*

Proof. Given in [Uni13, Theorem 4.2.3]. □

Example 2.94. Every involution $f : A \to A$ is half-adjoint to itself in an equivalence $f : A \simeq A$. For example, the identity function $\mathsf{id}_A : A \to A$ is an equivalence.

Example 2.95. We can produce an equivalence $\mathbf{1} + \mathbf{1} \simeq \mathbf{2}$ in the following way:

- a function $f : \mathbf{1} + \mathbf{1} \to \mathbf{2}$ is defined using the elimination principles of the coproduct and of the unit, by defining $f(\mathsf{inl}(*)) :\equiv \mathsf{yes}$ and $f(\mathsf{inr}(*)) :\equiv \mathsf{no}$;
- conversely, we can use $\mathsf{rec}_{\mathbf{2}}$ to define a function $g : \mathbf{2} \to \mathbf{1} + \mathbf{1}$, so that $g(\mathsf{yes}) :\equiv \mathsf{inl}(*)$ and $g(\mathsf{no}) :\equiv \mathsf{inr}(*)$;
- a dependent function $h_1 : \prod(b : \mathbf{2}) . f(g(b)) = \mathsf{id}_{\mathbf{2}}(b)$ is also defined by the elimination principle of $\mathbf{2}$ with $h_1(\mathsf{yes}) :\equiv \mathsf{refl}_{\mathsf{yes}}$ and $h_1(\mathsf{no}) :\equiv \mathsf{refl}_{\mathsf{no}}$;
- similarly for a dependent function $h_2 : \prod(x : \mathbf{1} + \mathbf{1}) . g(f(x)) = x$, defined by declaring $h_2(\mathsf{inl}(*)) :\equiv \mathsf{refl}_{\mathsf{inl}(*)}$ and $h_2(\mathsf{inr}(*)) :\equiv \mathsf{refl}_{\mathsf{inr}(*)}$.

Lemma 2.96. *For any type X, a function $f : X \to \mathbf{0}$ guarantees that X is equivalent to the empty type $\mathbf{0}$.*

Proof. The half-adjoint functions are f and $\mathsf{rec}_{\mathbf{0}} : \mathbf{0} \to X$; instances of $\mathsf{ind}_{\mathbf{0}}$ prove both that $f(\mathsf{rec}_{\mathbf{0}}(z)) = z$ for every $z : \mathbf{0}$ and that $\mathsf{rec}_{\mathbf{0}}(f(x)) = x$ for every $x : X$ (termwise eliminating into the family $(z \mapsto \mathsf{rec}_{\mathbf{0}}(z) = x)$). □

Lemma 2.97. *Any function between contractible types is an equivalence; any two contractible types are equivalent.*

Proof. If $f : A \to B$ is a function between contractible types, let its half-adjoint be defined as the function $B \to A$ constant at the center of contraction of A. Then the proofs of contractions of A and B provide homotopies $g \circ f \sim \mathsf{id}_A$ and $f \circ g \sim \mathsf{id}_B$. Since a function between contractible types can be always defined (constant at the center of the contraction), the second claim follows. □

Remark 2.98. A composition $(- \circ -) : (B \simeq C) \to (A \simeq B) \to (A \simeq C)$ of equivalences can be defined, for every A, B and $C : \mathcal{U}$; this is such that the function underlying a composite equivalence $g \circ f : A \simeq C$ is (judgmentally) the composition $g \circ f : A \to C$ of the functions underlying the equivalences $g : B \simeq C$ and $f : A \simeq B$.

Remark 2.99. For A, $A' : \mathcal{U}$, $B : A \to \mathcal{U}$ and $B' : A' \to \mathcal{U}$, given $f : A \simeq A'$ and $g : \prod(x : A) . B(x) \simeq B'(f(x))$, the function $\langle f, g \rangle$ in Definition 2.18 is an equivalence; similarly, functions $f \times g$ and $f + g$ (Definition 2.40) are equivalences whenever f and g are.

We immediately use Remark 2.99 in the following definition.

Definition 2.100. For $A, B : \mathcal{U}$, the function $\mathsf{incr} : (A \simeq B) \to ((A + \mathbf{1}) \simeq (B + \mathbf{1}))$ is defined by $\mathsf{incr}(e) :\equiv e + \mathsf{id}_{\mathbf{1}}$. We then have $\mathsf{incr}(e)\ (\mathsf{inl}(a)) \equiv \mathsf{inl}(e(a))$ for every $a : A$, and $\mathsf{incr}(e)\ (\mathsf{inr}(*)) \equiv \mathsf{inr}(*)$.

Remark 2.101. The operations described in Definitions 2.48 and 2.60 induce equivalences on identity types. For example, for every $X : \mathcal{U}$ and $x, y : X$, we have

$$\langle (p \mapsto p^{-1}), (p \mapsto p^{-1}), \mathsf{inv\_inv}, \mathsf{inv\_inv}, \ldots \rangle : (x = y) \simeq (y = x),$$

where $\mathsf{inv\_inv}$ is constructed in Definition 2.48(iii) and the proof of adjunction is given by path induction. Similarly, there is an equivalence

$$(x = z) \simeq \Sigma\,(y : X)\,.\,((x = y) \times (y = z))$$

induced by the concatenation, and an equivalence $(p = q) \simeq (p \cdot \mathsf{refl}_y = q)$ for every $p, q : x = y$, and so on. In the following lemma we will examine the equivalence relative to Definition 2.60(xi).

Lemma 2.102. *Let A and B be types and let $f, g : A \to B$. Let $p : x =_A y$ be a path in A. Consider the family $C : A \to \mathcal{U}$ defined as $C :\equiv (z \mapsto (f(z) = g(z)))$ and let $q_x : C(x)$. There is a* 2*-path*

$$p^C_*(q_x) = ([f](p))^{-1} \cdot q_x \cdot [g](p) \tag{2.103}$$

determining, for every $q_y : C(y)$, a corresponding equivalence:

$$(p^C_*(q_x) = q_y) \ \simeq\ (q_x \cdot [g](p) =_{(f(x)=g(y))} [f](p) \cdot q_y). \tag{2.104}$$

Proof. By induction on p, the 2-path in (2.103) is trivially obtained. As for the equivalence in (2.104), again by path induction, it is enough to verify that there is an equivalence

$$(q_x = q_y) \simeq (q_x \cdot \mathsf{refl}_{g(x)} = \mathsf{refl}_{f(x)} \cdot q_y);$$

which is obtained by means of path algebra (Remark 2.101). □

Remark 2.105. A consequence of Lemma 2.102 is that every homotopy $h : f \sim g$ determines a 2-path in B

$$h(x) \cdot [g](p) = [f](p) \cdot h(y) \tag{2.106}$$

for every $p : x =_A y$, as there is a path $[h]^{\mathsf{d}}(p) : p^C_*(h(x)) = h(y)$. In Chapter 3 we will use the notion of functions between types to represent functors between categories; the family of 2-paths in (2.106) then makes homotopies suited to represent natural isomorphisms between such functors.

Lemma 2.107. *The operation of transport is an equivalence, i.e., for every $A : \mathcal{U}$ and path $p : x =_A y$, the function $p_*^C : C(x) \to C(y)$ is an equivalence for any family $C : A \to \mathcal{U}$.*

Proof. The half-adjoint is $(p^{-1})_*^C : C(y) \to C(x)$; induction on p trivially provides the proof of equivalence. □

We will now focus on equivalences whose underlying function is the functorial action of a function on paths.

Definition 2.108 (Embeddings)**.** A function $f : A \to B$ is said to be an **embedding** if, for every $x, y : A$, the function $[f] : (x =_A y) \to (f(x) =_B f(y))$ is an equivalence, i.e., if there is a term in the type

$$\mathsf{IsEmb}(f) :\equiv \Pi\,(x, y : A)\,.\,\mathsf{IsEquiv}([f]),$$

where the arguments x and y are implicit in $[f]$.

If a function is an equivalence, then it is also an embedding, as shown below.

Lemma 2.109. *For every function $f : A \to B$, there is a function $\mathsf{IsEquiv}(f) \to \mathsf{IsEmb}(f)$.*

Proof. Let $\langle f^{-1}, h_1, h_2, \delta\rangle : \mathsf{IsEquiv}(f)$. For every $x, y : A$, we can define a function $\mathsf{unap}_f : (f(x) = f(y)) \to (x = y)$ by declaring

$$\mathsf{unap}_f(p) :\equiv (h_2(x))^{-1} \cdot [f^{-1}](p) \cdot h_2(y)$$

for every p. We have $[f] \circ \mathsf{unap}_f \sim \mathsf{id}_{(f(x)=f(y))}$, since, for every $p : f(x) = f(y)$,

$$\begin{aligned}
[f](\mathsf{unap}_f(p)) &= ([f](h_2(x)))^{-1} \cdot [f \circ f^{-1}](p) \cdot [f](h_2(y)) && \text{by path algebra} \\
&= (h_1(f(x)))^{-1} \cdot [f \circ f^{-1}](p) \cdot h_1(f(y)) && \text{by } \delta(y) \\
&= (h_1(f(x)))^{-1} \cdot h_1(f(x)) \cdot p = p && \text{by Remark 2.105.}
\end{aligned}$$

Moreover, it is easily shown that $\mathsf{unap}_f \circ [f] \sim \mathsf{id}_{(x=y)}$ by path induction, so $[f]$ is an equivalence, i.e., f is an embedding. □

Lemma 2.110. *Let $f : A \to B$ be an embedding. Then, for every $b : B$, the type $\mathsf{fib}_f(b)$ is a (-1)-type.*

Proof. By the elimination principle of Σ-types, for every $x, y : A$, $p : f(x) = b$ and $q : f(y) = b$ we need to find a path $\langle x, p\rangle =_{\mathsf{fib}_f(b)} \langle y, q\rangle$, i.e., by Lemma 2.66, a path $r : x = y$ and a 2-path $r_*^{(z\mapsto(f(z)=b))}(p) = q$. The first is defined as

$$r :\equiv \mathsf{unap}_f(p \cdot q^{-1}),$$

where unap_f is the inverse of $[f]$; the latter, by Lemma 2.102, entails finding a 2-path

$$p = [f](\mathsf{unap}_f(p \cdot q^{-1})) \cdot q,$$

which is given by path algebra and the fact that $[f]$ and unap_f are half-adjoint inverses. □

Univalence and Function Extensionality

For types $A, B : \mathcal{U}$, we have now established two notions of "identification": one is given by the identity type $A =_{\mathcal{U}} B$, while another one is given by the equivalence type $A \simeq B$. Voevodsky's *univalence axiom*, presented in the following paragraph, establishes a correspondence between these two notions and responds to the need of "modularity" principles in HoTT, namely, function extensionality and transport of structure [see e.g. Coq17].

Definition 2.111 (Univalence)**.** For every $A, B : \mathcal{U}$, consider the function

$$(p \mapsto p_*^{\mathsf{id}_{\mathcal{U}}}) : (A = B) \to (A \simeq B), \tag{2.112}$$

where Lemma 2.107 ensures that the transport is an equivalence. The **univalence axiom**, which we will assume in several parts of this thesis, states the following:

The function in (2.112) is an equivalence.

For convenience, will denote by $\mathsf{ua} : (A \simeq B) \to (A = B)$ the inverse of the function in (2.112), and by ua^{-1} the function itself.

Lemma 2.113. *The function* ua *respects identity, composition and inverses in the following way: we can construct* 2*-paths*

$$\mathsf{ua}(\mathsf{id}_A) = \mathsf{refl}_A, \qquad \mathsf{ua}(g \circ f) = \mathsf{ua}(f) \cdot \mathsf{ua}(g), \qquad \mathsf{ua}(f^{-1}) = (\mathsf{ua}(f))^{-1}.$$

Proof. The first two claims are proved explicitly in [Uni13, Section 2.10]; the third one, left to the reader, is proved here by way of example. For $A, B : \mathcal{U}$ and $f : A \simeq B$,

$$\begin{aligned}\mathsf{ua}(f^{-1}) &= \mathsf{ua}\left(\left(\mathsf{ua}^{-1}(\mathsf{ua}(f))\right)^{-1}\right) \equiv \mathsf{ua}\left(\left((\mathsf{ua}(f))_*^{\mathsf{id}_{\mathcal{U}}}\right)^{-1}\right) \\ &\equiv \mathsf{ua}\left(\left((\mathsf{ua}(f))^{-1}\right)_*^{\mathsf{id}_{\mathcal{U}}}\right) \equiv \mathsf{ua}\left(\mathsf{ua}^{-1}\left((\mathsf{ua}(f))^{-1}\right)\right) = (\mathsf{ua}(f))^{-1},\end{aligned}$$

using the fact that ua is an equivalence, the definition of ua^{-1} and the definition of the inverse of the transport equivalence, given in Lemma 2.107. □

Definition 2.114 (Function extensionality)**.** The principle of **function extensionality** states the following:

The function $[-]^*$ in Definition 2.84 is an equivalence.

In particular, for every $A : \mathcal{U}$, $B : A \to \mathcal{U}$ and $f, g : \Pi\,(a : A)\,.\,B(a)$, there is a function

$$\mathsf{fxt} : (f \sim g) \to (f = g)$$

such that $\mathsf{fxt}([-]^*) \sim \mathsf{id}_{(f=g)}$ and $[\mathsf{fxt}(-)]^* \sim \mathsf{id}_{(f\sim g)}$. It follows from the definition that there is a path

$$[-(a)](\mathsf{fxt}(h)) \equiv [\mathsf{fxt}(h)]^*(a) = h(a) \tag{2.115}$$

for every $h : f \sim g$ and $a : A$.

Like the univalence axiom, function extensionality cannot be deduced from the theory; we will make clear in this thesis the instances in which it is used. In particular, every time we will assume the univalence axiom, we will implicitly grant ourselves the right to use function extensionality, by virtue of the following lemma.

Lemma 2.116. *Assuming the univalence axiom, the principle of function extensionality can be derived.*

Proof. This is shown in [Uni13, Section 4.9]. □

Lemma 2.117. *The function* fxt *respects identity, composition and inverses in the following way: we can construct* 2*-paths*

$$\mathsf{fxt}(\mathsf{id}_f^*) = \mathsf{refl}_f \qquad \mathsf{fxt}(h \cdot^* h') = \mathsf{fxt}(h) \cdot \mathsf{fxt}(h') \qquad \mathsf{fxt}(h^{-1*}) = (\mathsf{fxt}(h))^{-1}.$$

Proof. This is shown using the fact that fxt is an equivalence, the definition of $[-]^*$, and Lemma 2.85. For example, we have:

$$\mathsf{fxt}(h \cdot^* h') = \mathsf{fxt}([\mathsf{fxt}(h)]^* \cdot^* [\mathsf{fxt}(h)]^*) = \mathsf{fxt}([\mathsf{fxt}(h) \cdot \mathsf{fxt}(h')]^*) = \mathsf{fxt}(h) \cdot \mathsf{fxt}(h'). \quad \square$$

Function extensionality also respects application of functions; in this thesis we will use the following lemma.

Lemma 2.118. *Given types $A, B, C : \mathcal{U}$, functions $f, g : A \to B$, $j : A \to A$ and $k : B \to C$ and a homotopy $h : f \sim g$, a* 2*-path*

$$[(z : A \to B) \mapsto (x : A) \mapsto k(z(x))](\mathsf{fxt}(h \circ j)) = \mathsf{fxt}(x \mapsto [k](h(j(x)))) \tag{2.119}$$

is defined.

Proof. As the function $[-]^*$ is an equivalence, there is a 2-path from the left-hand side of (2.119) to the path

$$\mathsf{fxt}([[(z : A \to B) \mapsto (x : A) \mapsto k(z(x))](\mathsf{fxt}(h \circ j))]^*) \ : (k \circ f \circ j) = (k \circ g \circ j),$$

so it is sufficient to find a 2-path

$$[[(z : A \to B) \mapsto (x : A) \mapsto k(z(x))](\mathsf{fxt}(h \circ j))]^* = (x \mapsto [k](h(j(x)))).$$

By function extensionality, it is enough to find a 2-path

$$[[(z : A \to B) \mapsto (x : A) \mapsto k(z(x))](\mathsf{fxt}(h \circ j))]^*(a) = [k](h(j(a)))$$

for every $a : A$. By Lemma 2.87 (on the left-hand side), we only need to find a 2-path

$$[k]([\mathsf{fxt}(h \circ j)]^*(a)) = [k](h(j(a))),$$

for every $a : A$, for which a 2-path $[\mathsf{fxt}(h \circ j)]^* = h \circ j$ suffices; this is given by the fact that $[-]^*$ and fxt are half-adjoint in an equivalence. □

Function extensionality allows us to give alternative characterizations of types of equivalences.

Lemma 2.120. *Assuming function extensionality, the following holds for $A, B : \mathcal{U}$:*

(i) for every $f : A \to B$, the type $\mathsf{IsEquiv}(f)$ is a (-1)-type;

(ii) for every $f : A \to B$, there is an equivalence

$$\mathsf{IsEquiv}(f) \simeq \Pi\,(b : B)\,.\ \mathsf{IsContr}(\mathsf{fib}_f(b)), \tag{2.121}$$

i.e., a function is an equivalence if and only if all its fibers are contractible;

(iii) given $e, e' : A \simeq B$, there is an equivalence

$$(e \sim e') \simeq (e =_{(A \simeq B)} e'), \tag{2.122}$$

where the homotopy is between functions underlying the equivalences.

Proof. The claims in (i) and (ii) are proved in [Uni13, Theorem 4.2.13 and Theorem 4.4.5]. As for (iii), given $h : e \sim e'$, function extensionality provides a path $\mathsf{fxt}(h) : e =_{(A \to B)} e'$, which by (i) and Remark 2.78 is enough to obtain a path $e =_{(\Sigma(f : A \to B).\, \mathsf{IsEquiv}(f))} e'$; the other direction and the proof of equivalence follow easily. □

It follows from Lemma 2.120(i) and Remark 2.75 that the type $\mathsf{IsEmb}(f)$ is a (-1)-type for every $f : A \to B$.

The following lemmata will be useful in Chapter 5 when dealing with the combinatorics of finite types.

Lemma 2.123. *Let A, A', B, $B' : \mathcal{U}$ and let $e : (A + B) \simeq (A' + B')$. Referring to Definition 2.41, given terms*

$$h_A : \Pi\,(a : A)\,.\,\mathsf{is\_inl}(e(\mathsf{inl}(a))), \qquad h_B : \Pi\,(b : B)\,.\,\mathsf{is\_inr}(e(\mathsf{inr}(b))),$$

and assuming function extensionality, there are equivalences $e|_A : A \simeq A'$ and $e|_B : B \simeq B'$ such that $e|_A + e|_B = e$. These will be called the restrictions *of e, respectively, to A and to B.*

Proof. The underlying function of $e|_A$ is $(a \mapsto \mathsf{inl}^{-1}(e(\mathsf{inl}(a)), h_A(a)))$ and that of $e|_B$ is defined similarly; for the proof that $e|_A$ and $e|_B$ are equivalences, we refer to the HoTT library [Hoq, `equiv_unfunctor_sum_l`, `equiv_unfunctor_sum_r`], where we conveniently found these results already proven. By Lemma 2.120(iii), we can show that $e|_A + e|_B = e$ by verifying that $\Pi\,(x : A + B)\,.\,(e|_A + e|_B)(x) = e(x)$, which we can do by induction on x. For every $a : A$, we have:

$$(e|_A + e|_B)(\mathsf{inl}(a)) \equiv \mathsf{inl}(e|_A(a)) \equiv \mathsf{inl}(\mathsf{inl}^{-1}(e(\mathsf{inl}(a)), h_A(a)))) = e(\mathsf{inl}(a))$$

by Lemma 2.71, and similarly for the other inductive case. □

Corollary 2.124. *Let $A, B : \mathcal{U}$ and $e : (A + \mathbf{1}) \simeq (B + \mathbf{1})$. Assuming function extensionality, if $e(\mathsf{inr}(*)) = \mathsf{inr}(*)$, then $e|_A : A \simeq B$ can be defined, and $\mathsf{incr}(e|_A) = e$, with incr as in Definition 2.100.*

Proof. Given $p : e(\mathsf{inr}(*)) = \mathsf{inr}(*)$, we obtain a term

$$(p^{-1})_*^{\mathsf{is\_inr}}(*) : \mathsf{is\_inr}(e(\mathsf{inr}(*))),$$

so, by the elimination principle of the unit type, we get a term

$$h_{\mathbf{1}} : \Pi\,(x : \mathbf{1})\,.\,\mathsf{is\_inr}(e(\mathsf{inr}(x))).$$

A term

$$h'_A : \Pi\,(a : A)\,.\,\Pi\,(x : B + \mathbf{1})\,.\,(e(\mathsf{inl}(a)) = x) \to \mathsf{is\_inl}(x)$$

is defined by induction on the term x, where $h'_A(a, \mathsf{inl}(b), q) :\equiv * : \mathbf{1} \equiv \mathsf{is\_inl}(\mathsf{inl}(b))$, and $h'_A(a, \mathsf{inr}(*), q)$ is given *ex falso* by Lemma 2.70, as there is a path

$$\begin{aligned} \mathsf{inl}(a) &= e^{-1}(e(\mathsf{inl}(a))) && \text{by the equivalence} \\ &= e^{-1}(e(\mathsf{inr}(*))) && \text{by } [e^{-1}](q \cdot p^{-1}) \\ &= \mathsf{inr}(*) && \text{by the equivalence.} \end{aligned}$$

Then $h_A : \Pi\,(a : A)\,.\,\mathsf{is\_inl}(e(\mathsf{inl}(a)))$ is defined as $h_A(a) :\equiv h'_A(a, e(\mathsf{inl}(a)), \mathsf{refl}_{e(\mathsf{inl}(a))})$. By Lemma 2.123, equivalences $e|_A : A \simeq B$ and $e|_{\mathbf{1}} : \mathbf{1} \simeq \mathbf{1}$ are defined, with $e = e|_A + e|_{\mathbf{1}}$ and $e|_{\mathbf{1}} \sim \mathsf{id}_{\mathbf{1}}$ (by the elimination principle of the unit type). Lemma 2.120(iii) gives a path $e|_{\mathbf{1}} = \mathsf{id}_{\mathbf{1}}$ between equivalences, so $\mathsf{incr}(e|_A) \equiv e|_A + \mathsf{id}_{\mathbf{1}} = e|_A + e|_{\mathbf{1}} = e$. □

Lemma 2.125. *Let $e : A \simeq B$ be an equivalence of types. Assuming function extensionality, there is an equivalence $((a = a') \to \mathbf{0}) \simeq ((e(a) = e(a')) \to \mathbf{0})$ for every $a, a' : A$.*

Proof. Similar to the proof of Lemma 2.109. □

Moreover, assuming function extensionality, we have the following result about truncation levels of function types.

Lemma 2.126. *Let $A : \mathcal{U}$ and $B : A \to \mathcal{U}$. If $B(a)$ is an n-type for every $a : A$, then so is the type $\Pi\,(a : A)\,.\,B(a)$ of dependent functions. Similarly, if $B : \mathcal{U}$ is an n-type, the type $A \to B$ of functions is also.*

Proof. This is shown in [Uni13, Examples 3.1.6 and 3.6.2] for (-1)- and 0-types; the proof for a general truncation level can be performed by induction on n. □

The function ua does not possess computational properties, as it is merely assumed as half-adjoint of another function in an equivalence. However, as we have seen in Lemma 2.113, the functorial action of ua on paths allows us sometimes to establish identities between paths in the universe $\mathcal{U}$ obtained via univalence, if corresponding paths between equivalences can be provided. Moreover, path induction techniques can be employed on paths of the form $\mathsf{ua}(e)$, where e is an equivalence of types; an example is given in the proof of the following lemma.

Lemma 2.127. *Let $A, B : \mathcal{U}$. For every $e : A \simeq B$, there is a path*

$$[\mathsf{add}](\mathsf{ua}(e)) =_{(A+\mathbf{1}=B+\mathbf{1})} \mathsf{ua}(\mathsf{incr}(e)), \tag{2.128}$$

defined in the proof, with add *and* incr *as in Definitions 2.42 and 2.100. Similarly, for every $p : A = B$, there is a path*

$$\mathsf{ua}^{-1}([\mathsf{add}](p)) =_{(A+\mathbf{1}\simeq B+\mathbf{1})} \mathsf{incr}(\mathsf{ua}^{-1}(p)), \tag{2.129}$$

as shown in Fig. 2.7.

Proof. We will prove (2.128), as (2.129) follows easily. Since ua is an equivalence, there is a path $\mathsf{ua}(\mathsf{incr}(\mathsf{ua}^{-1}(\mathsf{ua}(e)))) = \mathsf{ua}(\mathsf{incr}(e))$. A path

$$[\mathsf{add}](\mathsf{ua}(e)) = \mathsf{ua}(\mathsf{incr}(\mathsf{ua}^{-1}(\mathsf{ua}(e))))$$

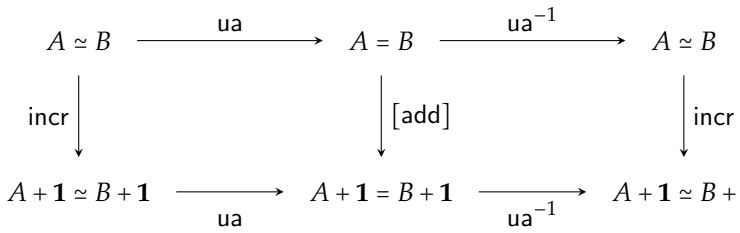


Figure 2.7: Relationship between incr and [add]. The two diagrams commute; on the left is (2.128), while on the right we find (2.129).

can be found by induction on $\mathsf{ua}(e) : A = B$, by providing a path

$$\mathsf{refl}_{A+\mathbf{1}} \equiv [\mathsf{add}](\mathsf{refl}_A) = \mathsf{ua}(\mathsf{incr}(\mathsf{ua}^{-1}(\mathsf{refl}_A))) \equiv \mathsf{ua}(\mathsf{incr}(\mathsf{id}_A)).$$

By ind_+, we can find a homotopy $\mathsf{id}_{A+\mathbf{1}} \sim \mathsf{incr}(\mathsf{id}_A)$; hence, by (2.122), there is a path $\mathsf{id}_{A+\mathbf{1}} =_{((A+\mathbf{1})\simeq(A+\mathbf{1}))} \mathsf{incr}(\mathsf{id}_A)$. Applying ua and using Lemma 2.113, a path

$$\mathsf{refl}_{A+\mathbf{1}} = \mathsf{ua}(\mathsf{id}_{A+\mathbf{1}}) = \mathsf{ua}(\mathsf{incr}(\mathsf{id}_A))$$

is obtained. □

We will conclude this section with the following lemma.

Lemma 2.130. *Let $A, B : \mathcal{U}$. Assuming univalence, if B is a (-1)-type, then so is $(A = B)$.*

Proof. Proved e.g. in [Hoq, `trunc_path_IsHProp`]. □

2.6 Higher Inductive Types

Higher inductive types (HITs) are a generalization of inductive types: they admit constructors both for terms in the type ("0-*constructors*") and for terms in the tower of its identity types ("*n-constructors*", if they are n-paths). Similarly to simple inductive types, the elimination principle of a HIT describes a property of "initiality" with respect to other families of types exhibiting terms and n-paths in correspondence to those provided by such constructors.

Accounts of the theory of HITs are given e.g. in [Uni13, Chapter 6; LS19; Soj15]. Useful mathematical constructions can be formalized through HITs, such as cell complexes, suspensions, wedge sums, smash products, suspensions and, in general, homotopy pushouts. In this section, we will show some examples of HITs, using

the same syntax adopted in Section 2.3 for the presentation of inductive types, and dubbing them "n-HITs" if they are defined with at least one n-constructor, and no $(n+1)$-constructors except, potentially, an m-truncation with $m \geq n-1$ (which provides all possible $(m+1)$-paths).

Example 2.131 (Interval). The interval $\mathbb{I}$ is the 1-HIT presented by the constructors:

$$\mathbb{I} :\equiv i_0 : \mathbb{I} \mid i_1 : \mathbb{I} \mid \mathsf{seg} : i_0 =_{\mathbb{I}} i_1.$$

When providing such a definition, the elimination principle we imply is the following. Given a family $C : \mathbb{I} \to \mathcal{U}$, a section $\mathsf{ind}_{\mathbb{I}} : \Pi\,(i : \mathbb{I})\,.\,C(i)$ is obtained by providing:

- terms $i'_0 : C(i_0)$ and $i'_1 : C(i_1)$;
- a pathover $\mathsf{seg}' : \mathsf{seg}^C_*(i'_0) =_{C(i_1)} i'_1$.

Note that the apparent asymmetry conveyed by the requirement seg' (i.e., a path in the fiber of the *second* endpoint of seg) is resolved by Lemma 2.107, by which we find a corresponding path in the fiber of i_0. The resulting elimination principle

$$\mathsf{ind}_{\mathbb{I}} : \Pi\left(i'_0 : C(i_0)\right).\Pi\left(i'_1 : C(i_1)\right).\left(\left(\mathsf{seg}^C_*(i'_0) = i'_1\right) \to \Pi\,(i : \mathbb{I})\,.\,C(i)\right)$$

computes on 0-constructors:

$$\mathsf{ind}_{\mathbb{I}}(i'_0, i'_1, \mathsf{seg}')(i_0) \equiv i'_0, \qquad \mathsf{ind}_{\mathbb{I}}(i'_0, i'_1, \mathsf{seg}')(i_1) \equiv i'_1.$$

Moreover, a 2-path

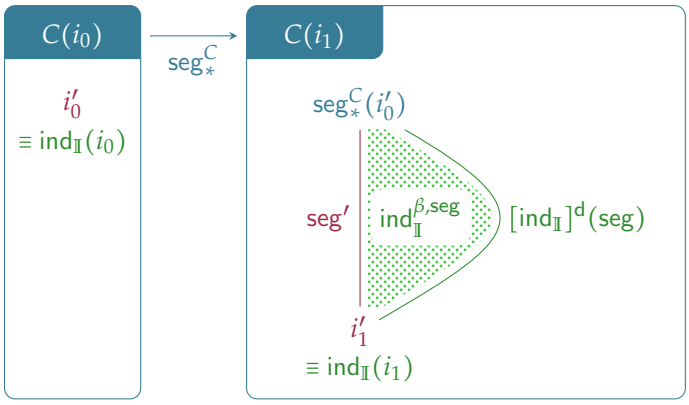
$$\mathsf{ind}^{\beta,\mathsf{seg}}_{\mathbb{I}} : [\mathsf{ind}_{\mathbb{I}}(i'_0, i'_1, \mathsf{seg}')]^{\mathsf{d}}(\mathsf{seg}) = \mathsf{seg}'$$

is assumed as an axiom; although the two sides of the identity are not judgmentally equal [see the discussion in Uni13, Section 6.2], we will still refer to the axiom $\mathsf{ind}^{\beta,\mathsf{seg}}_{\mathbb{I}}$ as a computation rule of the elimination principle. The workings of $\mathsf{ind}_{\mathbb{I}}$ is shown in Fig. 2.8.

It is useful to observe that, when eliminating into a family of identity types, we can specialise the elimination principle by using Lemma 2.102 to simplify the requirements. That is, given a type B and functions $f, g : \mathbb{I} \to B$, a section $\mathsf{ind}_{\mathbb{I}} : \Pi\,(i : \mathbb{I})\,.\,C(i)$ for the family $C :\equiv (z \mapsto f(z) = g(z))$ can be obtained by providing paths $i'_0 : f(i_0) = g(i_0)$ and $i'_1 : f(i_1) = g(i_1)$, and a 2-path $\mathsf{seg}' : i'_0 \cdot [g](\mathsf{seg}) = [f](\mathsf{seg}) \cdot i'_1$ *in lieu* of the 2-path specified above.

If T is a type, the non-dependent version $\mathsf{rec}_{\mathbb{I}} : \mathbb{I} \to T$ of the elimination principle can be derived by $\mathsf{ind}_{\mathbb{I}}$ assuming the family C constant at T; that is, there is a function

$$\mathsf{rec}_{\mathbb{I}} : \Pi\left(i'_0, i'_1 : T\right).(i'_0 = i'_1) \to \mathbb{I} \to T$$



$$\mathbb{I} :\equiv \quad i_0 \overset{\mathsf{seg}}{\longrightarrow} i_1 \qquad \mathsf{ind}_{\mathbb{I}} : \Pi\left(i'_0, i'_1, \mathsf{seg}'\right).\Pi\left(x : \mathbb{I}\right).C(x)$$

Figure 2.8: The interval $\mathbb{I}$ as a HIT, with, in red, the terms required by the elimination principle $\mathsf{ind}_{\mathbb{I}}$ in order to produce a section $\Pi\,(x : \mathbb{I})\,.\,C(x)$ and, in green, its computation rules.

given by

$$\mathsf{rec}_{\mathbb{I}}(i'_0, i'_1, \mathsf{seg}') :\equiv \mathsf{ind}_{\mathbb{I}}(i'_0, i'_1,\ \mathsf{tr\_const}(\mathsf{seg}, i'_0) \cdot \mathsf{seg}'),$$

where $\mathsf{tr\_const}(\mathsf{seg}, i'_0) : \mathsf{seg}^{(z \mapsto T)}_*(i'_0) = i'_0$ is constructed as in Definition 2.60(ix). Such a function computes on i_0 and i_1:

$$\mathsf{rec}_{\mathbb{I}}(i'_0, i'_1, \mathsf{seg}')(i_0) \equiv i'_0, \qquad \mathsf{rec}_{\mathbb{I}}(i'_0, i'_1, \mathsf{seg}')(i_1) \equiv i'_1.$$

In addition, a 2-path $\mathsf{rec}^{\beta,\mathsf{seg}}_{\mathbb{I}} : [\mathsf{rec}_{\mathbb{I}}(i'_0, i'_1, \mathsf{seg}')](\mathsf{seg}) = \mathsf{seg}'$ can be derived; indeed, we have:

$$\begin{aligned}
&\mathsf{tr_const}(\mathsf{seg}, i'_0) \cdot [\mathsf{rec}_{\mathbb{I}}(i'_0, i'_1, \mathsf{seg}')](\mathsf{seg}) \\
&\quad = [\mathsf{rec}_{\mathbb{I}}(i'_0, i'_1, \mathsf{seg}')]^{\mathsf{d}}(\mathsf{seg}) && \text{by Definition 2.60(xiii)} \\
&\quad \equiv [\mathsf{ind}_{\mathbb{I}}(i'_0, i'_1,\ \mathsf{tr_const}(\mathsf{seg}, i'_0) \cdot \mathsf{seg}')]^{\mathsf{d}}(\mathsf{seg}) \\
&\quad = \mathsf{tr_const}(\mathsf{seg}, i'_0) \cdot \mathsf{seg}' && \text{by } \mathsf{ind}^{\beta,\mathsf{seg}}_{\mathbb{I}},
\end{aligned}$$

so the sought 2-path $[\mathsf{rec}_{\mathbb{I}}(i'_0, i'_1, \mathsf{seg}')](\mathsf{seg}) = \mathsf{seg}'$ can be obtained by left cancelling (Definition 2.48(viii)).

Similarly to the interval, we can define the circle as a HIT.

Example 2.132 (Circle). The circle S^1 is the 1-HIT presented by the constructors:

$$\mathsf{S}^1 :\!:= \mathsf{base} : \mathsf{S}^1 \ \mid \mathsf{loop} : \mathsf{base} =_{\mathsf{S}^1} \mathsf{base}.$$

This presentation implies an elimination principle ind_{S^1}, its non-dependent version rec_{S^1} and their computation rules, specified analogously to those in Example 2.131 for the interval $\mathbb{I}$.

Simple HITs with 2-constructors (or higher) can be defined similarly, although spelling out the type of their elimination principles is more tedious. We present here a toy HIT representing a "filled" polygon; the same example can be instantiated to HITs representing more interesting shapes, such as the torus, the sphere, etc.

Example 2.133 (Filled triangle). The filled triangle $\blacktriangleright$ is the 2-HIT presented by the constructors:

$$\blacktriangleright :\equiv a : \blacktriangleright \mid b : \blacktriangleright \mid c : \blacktriangleright \mid p : a =_{\blacktriangleright} b \mid q : b =_{\blacktriangleright} c \mid r : a =_{\blacktriangleright} c \mid s : p \cdot q =_{(a=c)} r.$$

Given a family $C : \blacktriangleright \to \mathcal{U}$, a section $\mathsf{ind}_{\blacktriangleright} : \prod(x : \blacktriangleright).\, C(x)$ is obtained by providing:

- terms $a' : C(a)$, $b' : C(b)$ and $c' : C(c)$;
- pathovers
$$p' : p^C_*(a') =_{C(b)} b', \qquad q' : q^C_*(b') =_{C(c)} c' \qquad \text{and} \qquad r' : r^C_*(a') =_{C(c)} c';$$
- a 2-path $s' : ([(-)^C_*(a')](s))^{-1} \cdot (p' \cdot^{\mathsf{d}} q') =_{(r^C_*(a')=c')} r'$,

as shown in Fig. 2.9. Again, the ensuing function

$$\mathsf{ind}_{\blacktriangleright} : \prod\big(a' : C(a)\big).\, \prod\big(b', c', p', q', r', s' : \ldots\big).\, \prod(x : \blacktriangleright).\, C(x)$$

computes on 0-constructors:

$$\mathsf{ind}_{\blacktriangleright}(a', b', c', p', q', r', s')(a) \equiv a', \qquad \mathsf{ind}_{\blacktriangleright}(a', b', c', p', q', r', s')(b) \equiv b', \qquad \ldots,$$

and there are 2-paths

$$\mathsf{ind}^{\beta,p}_{\blacktriangleright} : [\mathsf{ind}_{\blacktriangleright}(a', b', c', p', q', r', s')]^{\mathsf{d}}(p) = p',$$

and similarly $\mathsf{ind}^{\beta,q}_{\blacktriangleright}$ and $\mathsf{ind}^{\beta,r}_{\blacktriangleright}$, assumed axiomatically. Finally, a 3-path $\mathsf{ind}^{\beta,s}_{\blacktriangleright}$ is assumed between the 2-path

$$[[\mathsf{ind}_{\blacktriangleright}(\ldots)]^{\mathsf{d}}]^{\mathsf{d}}(s) : s_*^{(z:(a=c)\mapsto z^C_*(a')=c')}\Big([\mathsf{ind}_{\blacktriangleright}(\ldots)]^{\mathsf{d}}(p \cdot q)\Big) = [\mathsf{ind}_{\blacktriangleright}(\ldots)]^{\mathsf{d}}(r)$$

and the 2-path obtained as the concatenation:

$$\begin{aligned}
&s_*^{(z:(a=c)\mapsto z^C_*(a')=c')}\Big([\mathsf{ind}_{\blacktriangleright}(\ldots)]^{\mathsf{d}}(p \cdot q)\Big) \\
&\quad = ([(-)^C_*(a')](s))^{-1} \cdot [\mathsf{ind}_{\blacktriangleright}(\ldots)]^{\mathsf{d}}(p \cdot q) \\
&\quad = ([(-)^C_*(a')](s))^{-1} \cdot ([\mathsf{ind}_{\blacktriangleright}(\ldots)]^{\mathsf{d}}(p) \cdot^{\mathsf{d}} [\mathsf{ind}_{\blacktriangleright}(\ldots)]^{\mathsf{d}}(q)) \\
&\quad = ([(-)^C_*(a')](s))^{-1} \cdot (p' \cdot^{\mathsf{d}} q') \\
&\quad = r' = [\mathsf{ind}_{\blacktriangleright}(\ldots)]^{\mathsf{d}}(r),
\end{aligned}$$

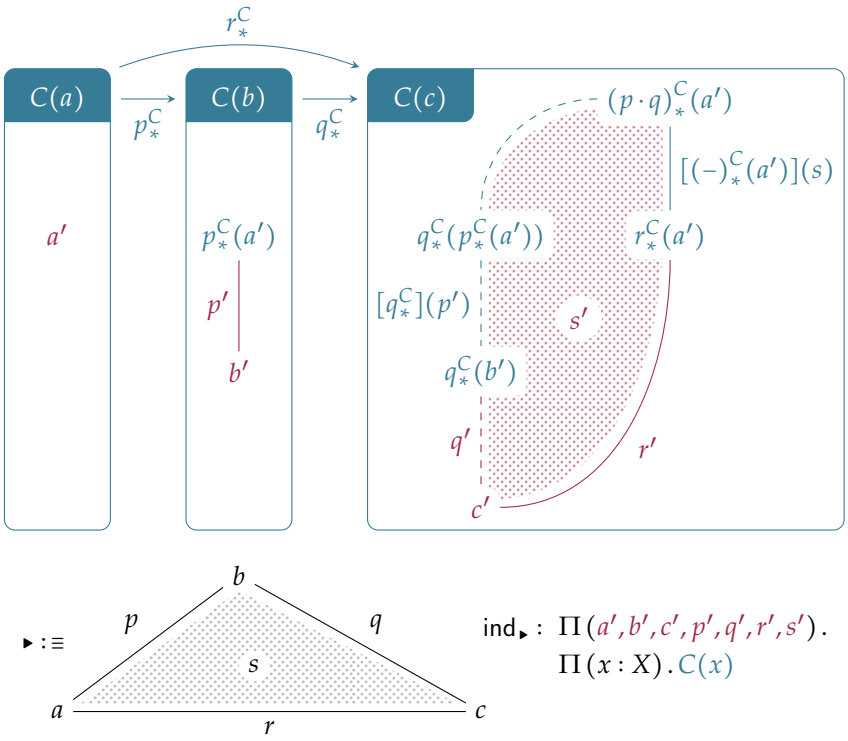


Figure 2.9: The filled triangle $\blacktriangleright$ as a HIT, with (in red) the terms required by the elimination principle $\mathsf{ind}_\blacktriangleright$ in order to produce a section $\Pi(x : \blacktriangleright).\, C(x)$. The dashed path, top-to-bottom, is $p' \cdot^{\mathrm{d}} q'$.

using path algebra lemmata, $\mathsf{ind}^{\beta,p}_\blacktriangleright$, $\mathsf{ind}^{\beta,q}_\blacktriangleright$, $\mathsf{ind}^{\beta,r}_\blacktriangleright$ and s'. We reiterate that, if C is a family of identity types, the discussion in Example 2.131 still applies, and the requirements can be given a simpler form.

The non-dependent version of the elimination principle, which can be derived from what above, states that a function $\mathsf{rec}_\blacktriangleright : \blacktriangleright \to T$ can be produced for any type T, once given terms $a', b', c' : T$, paths $p' : a' = b'$, $q' : b' = c'$ and $r' : a' = c'$, and a 2-path $s' : p' \cdot q' = r'$. Such a function computes on a, b and c to a', b' and c' respectively; there is a 2-path $\mathsf{rec}^{\beta,p}_\blacktriangleright : [\mathsf{rec}_\blacktriangleright](p) = p'$ and similarly for q and r; and a 3-path $\mathsf{rec}^{\beta,s}_\blacktriangleright$ between $[[\mathsf{rec}_\blacktriangleright]](s) : [\mathsf{rec}_\blacktriangleright](p \cdot q) = [\mathsf{rec}_\blacktriangleright](r)$ and the following one, obtained, via path algebra, using $\mathsf{rec}^{\beta,p}_\blacktriangleright$, $\mathsf{rec}^{\beta,q}_\blacktriangleright$, $\mathsf{rec}^{\beta,r}_\blacktriangleright$ and s':

$$[\mathsf{rec}_\blacktriangleright](p \cdot q) = [\mathsf{rec}_\blacktriangleright](p) \cdot [\mathsf{rec}_\blacktriangleright](q) = p' \cdot q' = r' = [\mathsf{rec}_\blacktriangleright](r).$$

Remark 2.134. Oftentimes we will be in the situation of applying the elimination principle of a HIT to families of n-types. If n is sufficiently small, some of the argu-

ments of the elimination principle are provided by the truncation level of the target type. For example, if $C : \blacktriangleright \to \mathcal{U}$ is a family of 0-types, we will not need to specify explicitly the term s' in Example 2.133, as the truncation level of $C(c)$ guarantees the presence of such a 2-path (which is unique, up to homotopy).

Of particular relevance to our thesis are *truncations* of types, which we present below. We refer to [Uni13] for a more in-depth exposition on the subject.

Definition 2.135 (Truncation)**.** Let n be a truncation level. The **n-truncation** $\|X\|_n$ of a type X is a HIT with the following constructors:

$$\|X\|_n ::= |-| : X \to \|X\|_n \mid T : \mathsf{IsTrunc}_n(\|X\|_n).$$

Here, $|-|$ is a 0-constructor, while T acts as an n-constructor for every greater n. Its elimination principle states that, for every family $C : \|X\|_n \to \mathcal{U}$, a function $\mathsf{ind}_{\|-\|} : \Pi\left(y : \|X\|_n\right). C(y)$ is obtained by exhibiting:

- a term $|-|' : \Pi\,(x : X)\,. C(|x|)$, and
- a term $T' : \Pi\,(x : X)\,.\ \mathsf{IsTrunc}_n(C(|x|))$;

such a function will compute $\mathsf{ind}_{\|-\|}(|x|) \equiv |x|'$ for all $x : X$. The non-dependent elimination principle $\mathsf{rec}_{\|-\|}$ is derived accordingly. The (-1)-truncation $\|X\|_{-1}$ of X is also denoted by $\|X\|$.

It follows from the definition that $\|X\|_n$ is an n-type.

Remark 2.136. The truncation of a type is our first example of a *recursive* HIT, i.e., of a HIT with a constructor quantifying over the terms (and paths) in the HIT itself. Indeed, we see, for example, that the constructor T in $\|X\|$ has type

$$T : \Pi\,(x, y : \|X\|)\,. x = y.$$

Recursive HITs are, in general, harder to work with than non-recursive ones; they fail to be encapsulated into a general pattern and they are not well supported by all the main proof assistants currently in use (such as Lean [vDvRB17]). Efforts were spent in order to replicate recursive HITs into non-recursive constructions [vD15; Kra16], although it is not known to which extent this is possible. An easy example is given by these two definitions of the half-line:

$$\mathbb{H} ::= \overline{0} : \mathbb{H} \mid \overline{\mathsf{s}} : \mathbb{H} \to \mathbb{H} \mid l : \Pi\,(x : \mathbb{H})\,. x = \overline{\mathsf{s}}(x) \qquad \text{(recursive)};$$

$$\mathbb{H}' ::= i : \mathbb{N} \to \mathbb{H}' \mid l : \Pi\,(n : \mathbb{N})\,. i(n) = i(n+1) \qquad \text{(non-recursive)},$$

where the recursion in $\mathbb{H}'$ is "hidden" in the *family* of constructors i, which is indexed by a recursive inductive type (the natural numbers); the two types are equivalent because they are both provably contractible (Lemma 2.97).

In this thesis, we will make extensive use of recursive HITs.

Remark 2.137. The (-1)-truncation of a type ("*propositional truncation*") is especially interesting: a term in $\|X\|$ provides a proof that X is inhabited in a *weaker* sense than exhibiting a term in X, since we cannot use the elimination principle of the truncation in order to define a function $\|X\| \to X$, unless X is a (-1)-type (we do remark that there are means to produce functions out of the (-1)-truncation of a type even if the target type is not a (-1)-type [Kra15]). However, the function

$$(f \mapsto x \mapsto f(|x|)) : (\|X\| \to \mathbf{0}) \to X \to \mathbf{0},$$

shows the contrapositive (but not equivalent) statement: if $\|X\|$ is *not* inhabited, then neither is X.

The (-2)-truncation of a type is, on the contrary, far less interesting, being it always equivalent to the unit type.

Remark 2.138 ("There is"). The propositional truncation offers a counterpart in HoTT to statements in classical (non proof-relevant) mathematics; for example, the statement "there exists an even natural number" could be properly transposed to the type $\|\Sigma\,(n : \mathbb{N})\,.\,E(n)\|$, for a suitable definition of the family $E : \mathbb{N} \to \mathcal{U}$. In this thesis, we adopt the convention by which, when claiming informally that "there is" a term in a certain type X (e.g. "there is an equivalence..."), we do *not* just mean that the propositional truncation of X is inhabited, but instead we mean that we provide a specific, explicit construction for a term in X, which will always be accessible.

Lemma 2.139. *Assuming univalence, for any $A : \mathcal{U}$, $a, b : A$ and any truncation level n, there is an equivalence*

$$\mathsf{path\_trunc} : (|a| =_{\|A\|_{n+1}} |b|) \simeq \|a =_A b\|_n\,. \tag{2.140}$$

Proof. We omit the details of the proof of this statement, which can be found in the HoTT library [Hoq, `equiv_path_Tr`]. The two types are shown equivalent via a standard "encode-decode" proof (which we will discuss in Section 3.4), characterizing the identity types in $\|A\|_{n+1}$; this is done by defining a function

$$f : \|A\|_{n+1} \to \|A\|_{n+1} \to \mathcal{U}$$

which is supposed to compute $f(|a|,|b|) \equiv \|a = b\|_n$ for every $a, b : A$ and to be such that $(x =_{\|A\|_{n+1}} y) \simeq f(x,y)$ for every $x, y : \|A\|_{n+1}$. However, the elimination

principle of the truncation cannot be employed directly in order to define f, as $\mathcal{U}$ is not a $(n+1)$-type. Hence, f is defined as $\mathsf{pr}_1 \circ f'$, with

$$f' : \|A\|_{n+1} \to \|A\|_{n+1} \to \Sigma\,(X : \mathcal{U})\,.\,\mathsf{IsTrunc}_n(X),$$

i.e., factoring through the subuniverse of n-types; univalence is used to prove that the latter is an $(n+1)$-type. □

We will make use, in Chapter 5, of the following notions.

Definition 2.141 (Surjectivity)**.** A function $f : A \to B$ is said to be **surjective** if the (-1)-truncation of the fiber over any term in B is inhabited, i.e., if one can provide a term in

$$\Pi\,(b : B)\,.\,\left\|\mathsf{fib}_f(b)\right\|.$$

Definition 2.142 (Connected types)**.** A type X is said to be **connected** if the type $\|X\|_0$ is contractible. If we assume univalence, Lemma 2.139 shows that, in order to prove that a type X is connected, it is enough to provide a term in $x : X$ (so that $|x| : \|X\|_0$ can be the center of contraction) and a term in $\Pi\,(a, b : X)\,.\,\|a = b\|$.

Connected types satisfy the following lemma.

Lemma 2.143. *Let A be a connected type and $C : A \to \mathcal{U}$ be a property (in the sense of Definition 2.73). Assuming univalence, if any term $a_0 : A$ satisfies the property C, then all terms in A do, i.e. there is a function*

$$\mathsf{conn\_to\_prop} : \Pi\,(a_0 : A)\,.\,(C(a_0) \to \Pi\,(a : A)\,.\,C(a)),$$

defined in the proof.

Proof. For every $a_0, a : A$ and $x : C(a_0)$, a function

$$f_{a_0,a,x} : \|a_0 = a\| \to P(a)$$

is obtained using the elimination principle of the truncation; as $P(a)$ is a (-1)-type by hypothesis, it is enough to define $f(|p|) :\equiv p^C_*(x)$, for any $p : a_0 = a$. Since A is connected, for any $a_0, a : A$ there is a path $q_{a_0,a} : |a_0| =_{\|A\|_0} |a|$ given by the proof of contraction of $\|A\|_0$, so we can define

$$\mathsf{conn\_to\_prop}(a_0, x, a) :\equiv f_{a_0,a,x}(\mathsf{path\_trunc}(q_{a_0,a}))$$

for every $x : C(a_0)$, with path_trunc as in Lemma 2.139. □

By adding the constructor T of the truncation to the definition of other HITs, we obtain (n-)*truncated* HITs; their elimination principles combine what seen above for truncations with the presented scheme for HIT-elimination. In many cases, we will not need to specify all the computation rules of the elimination principle of a truncated HIT: for example, a 1-truncated HIT can only eliminate into a family of 1-types, so it is redundant to make the computation rules relative to its 2-constructors (if any) explicit, as they would be 3-paths in a 1-type. Finitary 1-truncated HITs (i.e., with constructors having a finite number of recursive arguments) are discussed e.g. in [VvdW20].

An interesting kind of 2-constructor for a HIT is one that involves the interplay between 1-constructors and the application of a recursive 0-constructor to 1-constructors. We call the resulting HITs "**ap-recursive**"; an example showing the elimination scheme is provided below.

Example 2.144 (An ap-recursive HIT). Let R be the ap-recursive 2-HIT presented by the following constructors, and depicted in Fig. 2.10:

$$\mathsf{R} :\equiv z : \mathsf{R} \mid f : \mathsf{R} \to \mathsf{R} \mid p : \Pi\,(x : \mathsf{R})\,.\,x = f(x) \mid s : \Pi\,(x : \mathsf{R})\,.\,[f](p_x) = p_{f(x)}.$$

The definition of R is similar to the half-line $\mathbb{H}$ in Remark 2.136, but we included additional 2-paths. For a family $C : \mathsf{R} \to \mathcal{U}$ of types, the elimination principle of R states that a function $\mathsf{ind}_{\mathsf{R}} : \Pi\,(x : \mathsf{R})\,.\,C(x)$ is produced by providing:

- a term $z' : C(z)$;
- a function $f' : \Pi\,(x : \mathsf{R})\,.\,C(x) \to C(f(x))$;
- for every $x : \mathsf{R}$ and $y : C(x)$, a pathover $p'_x(y) : (p_x)^C_*(y) =_{C(f(x))} f'_x(y)$;

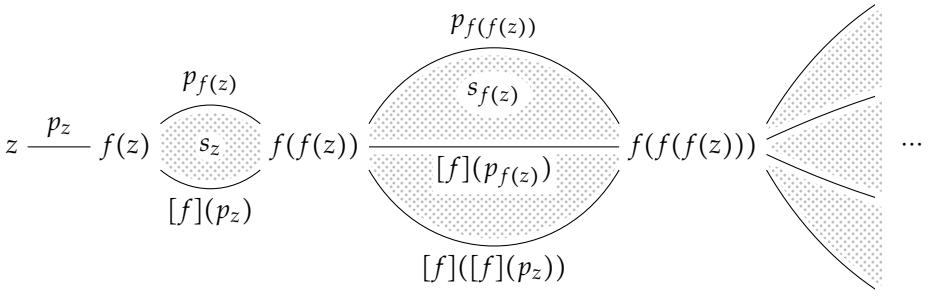


Figure 2.10: Some of the terms, paths and 2-paths in the ap-recursive 2-HIT R; the unmarked 2-path at the bottom center of the figure is given by $[[f]](s_z)$.

- for every $x : \mathsf{R}$ and $y : C(x)$, a 2-path

$$s'_x(y) : ([(-)^C_*(f'_x(y))](s_x))^{-1} \cdot \left({}^{f|f'}(p'_x(y))\right) = p'_{f(x)}(f'_x(y)),$$

  where the identity is between terms in $(p_{f(x)})^C_*(f'_x(y)) = f'_{f(x)}(f'_x(y))$, with ${}^{f|f'}(p'_x(y)) : ([f](p_x))^C_*(f'_x(y)) = f'_{f(x)}(f'_x(y))$ (Definition 2.60(x)).

The computation rules of $\mathsf{ind}_{\mathsf{R}}$ (and its non-dependent version $\mathsf{rec}_{\mathsf{R}}$) follow the scheme given in previous examples.

Indexed families of HITs can also be defined, with elimination principles following a pattern combining the one given above and the one shown in Section 2.3. In general, a HIT might present several of the attributes we presented. For example, in Chapter 5 we will define the deloopings of symmetric groups as an indexed family of ap-recursive, 1-truncated HITs. Other kinds of HITs exist, such as higher inductive-inductive types [KK18] and quotient inductive-inductive types [Alt+18]. They will not appear in this thesis, and hence they are not discussed here.

Chapter 3

Coherence for Monoidal Groupoids

One of the core issues we meet in this thesis is finding good definitions in HoTT of the objects of our study; that is, definitions that match our understanding of the mathematical concepts they are supposed to represent, while fitting in the general theory, and that make it feasible to prove theorems about them. The last point is crucial in a proof-relevant environment, as the same object can be designed (i.e., defined) in different but equivalent ways. In HoTT, equivalent objects share the same logical properties; however, the very design of a definition can make proofs of specific statements easier or harder to achieve. Before introducing the topic of coherence for monoidal groupoids, we will present an example to illustrate this phenomenon.

The content of this chapter appears in [Pic20] in a shortened form.

3.1 Motivation

Let X be a type and consider the type $\mathsf{list}(X)$ and the operation $+\!\!+$ of list append given in Definition 2.32. We saw in Remark 2.36 that $+\!\!+$ satisfies the left-unit law judgmentally with respect to nil, i.e., $\mathsf{nil} +\!\!+ l \equiv l$ for every $l : \mathsf{list}(X)$. Hence, we can easily construct a term

$$\lambda_{\mathsf{list}} : \Pi\,(l : \mathsf{list}(X))\,.\,\mathsf{nil} +\!\!+ l = l$$

by declaring $\lambda_{\mathsf{list}} :\equiv (l \mapsto \mathsf{refl}_l)$. Using the elimination principle of lists, we can also prove that $+\!\!+$ satisfies the right-unit law and the associative law, as shown in the following lemma.

Lemma 3.1. *We construct terms*

$$\rho_{\mathsf{list}} : \prod (l : \mathsf{list}(X)) . l +\!\!+ \mathsf{nil} = l \qquad \textit{and}$$
$$\alpha_{\mathsf{list}} : \prod (l_1,\ l_2,\ l_3 : \mathsf{list}(X)) . (l_1 +\!\!+ l_2) +\!\!+ l_3 = l_1 +\!\!+ l_2 +\!\!+ l_3.$$

Proof. We will define the term α_{list}, hence showing associativity of $+\!\!+$; right unitality is proved similarly. By induction on l_1, it is enough to define the terms

$$\begin{aligned}\alpha_{\mathsf{list}}(\mathsf{nil},\ l_2,\ l_3) &:\equiv \mathsf{refl}_{l_2 +\!\!+ l_3} \\ &: (\mathsf{nil} +\!\!+ l_2) +\!\!+ l_3 \equiv l_2 +\!\!+ l_3 = l_2 +\!\!+ l_3 \equiv \mathsf{nil} +\!\!+ l_2 +\!\!+ l_3\end{aligned}$$

and, given a term $\alpha_{\mathsf{list}}(l_1,\ l_2,\ l_3)$ for some l_1 (*inductive hypothesis*),

$$\begin{aligned}\alpha_{\mathsf{list}}(x :: l_1,\ l_2,\ l_3) &:\equiv [x :: -](\alpha_{\mathsf{list}}(l_1,\ l_2,\ l_3)) \\ &: (x :: l_1 +\!\!+ l_2) +\!\!+ l_3 \equiv x :: ((l_1 +\!\!+ l_2) +\!\!+ l_3) \\ &= x :: (l_1 +\!\!+ l_2 +\!\!+ l_3) \equiv x :: l_1 +\!\!+ l_2 +\!\!+ l_3\end{aligned}$$

for every $x : X$. □

Having a unit and a product satisfying associativity and unitality, we can think of $\mathsf{list}(X)$ as a *monoid*. It could be desirable to generalize this idea and specify a type of types with the structure of a monoid:

$$\begin{aligned}\mathsf{Mon} :\equiv \sum (M : \mathcal{U}) . \sum (e : M) . \sum (\otimes : M \to M \to M) . \\ (\prod (a,\ b,\ c : M) . (a \otimes b) \otimes c = a \otimes (b \otimes c)) \\ \times (\prod (a : M) . e \otimes b = b) \\ \times (\prod (a : M) . a \otimes e = a).\end{aligned}$$

The type Mon is a data structure consisting of a *carrier* M, a unit term e and a product $\otimes$ (here used with infix notation) satisfying the monoid laws, which we will denote, whenever needed, by the symbols α, λ and ρ (for associativity and left and right unitality, respectively).[1] For example, we have:

$$\langle \mathsf{list}(X), \mathsf{nil}, +\!\!+, \alpha_{\mathsf{list}}, \lambda_{\mathsf{list}}, \rho_{\mathsf{list}} \rangle : \mathsf{Mon}.$$

We will informally use the carrier to denote the tuple, and write e.g. $\mathsf{list}(X) : \mathsf{Mon}$. We stress that a term $M : \mathsf{Mon}$ is only a monoid (in the classical sense) if its carrier is a 0-type.

[1] Mon inhabits a higher universe than the one over which it quantifies (where the carriers live), but we will not be concerned with this detail.

For a function $f : M \to N$ between the carriers of two monoids, a reasonable question to ask is whether it preserves unit and product, i.e. whether it is the case that $f(e_M) = e_N$ and $f(a) \otimes_N f(b) = f(a \otimes_M b)$ for every $a, b : M$, thus gaining the status of *monoid homomorphism*. Given a type X, a monoid N and a function g from X to the carrier of N, we can always produce a function f_g from $\mathsf{list}(X)$ to the carrier of N preserving the monoid structure. Indeed, we can define:

$$f_g(\mathsf{nil}) :\equiv e_N, \qquad f_g(x :: l) :\equiv g(x) \otimes_N f_g(l). \tag{3.2}$$

From (3.2), we immediately see that f_g preserves the unit (judgmentally). In order to show that it also preserves the product, we need a term

$$f_g^{++} : \Pi\,(l_1,\ l_2 : \mathsf{list}(X))\,.\, f_g(l_1) \otimes_N f_g(l_2) = f_g(l_1 +\!\!+ l_2),$$

which we can obtain by induction. For every $l_2 : \mathsf{list}(X)$, we specify

$$\begin{aligned} f_g^{++}(\mathsf{nil}, l_2) &:\equiv \lambda_N(f_g(l_2)) \\ &: f_g(\mathsf{nil}) \otimes_N f_g(l_2) \equiv e_N \otimes_N f_g(l_2) = f_g(l_2) \equiv f_g(\mathsf{nil} +\!\!+ l_2) \end{aligned}$$

where λ_N is the left unitality for $\otimes_N$; and, given a term $f_g^{++}(l_1, l_2)$ for some l_1,

$$\begin{aligned} f_g^{++}(x :: l_1, l_2) &:\equiv \alpha_N(g(x), f_g(l_1), f_g(l_2)) \cdot [g(x) \otimes_N -](f_g^{++}(l_1, l_2)) \\ &: f_g(x :: l_1) \otimes_N f_g(l_2) \equiv (g(x) \otimes_N f_g(l_1)) \otimes_N f_g(l_2) \\ &= g(x) \otimes_N (f_g(l_1) \otimes_N f_g(l_2)) \\ &= g(x) \otimes_N f_g(l_1 +\!\!+ l_2) \equiv f_g(x :: l_1 +\!\!+ l_2) \end{aligned}$$

for every $x : X$, where α_N is the associativity for $\otimes_N$.

The complexity of the definition of f_g^{++} is due to the fact that the operation $+\!\!+$ is not a constructor of $\mathsf{list}(X)$. Instead, it is defined by list-elimination, and hence statements about $+\!\!+$ are likely to require unfolding its definition (and producing another proof by induction). Further statements involving f_g^{++} – for example, verifying that f_g also respects associativity – would entail unfolding *its* definition and reasoning on the paths λ_N and α_N appearing in each of the inductive cases. This particular issue occurs, for example, if we want to prove that $\mathsf{list}(X)$ is "the" *free* monoid over X, which entails finding an inverse to the function $(g \mapsto f_g)$ (here the obvious choice is the function sending a monoid homomorphism $h : \mathsf{list}(X) \to N$ to the function $h \circ (- :: \mathsf{nil}) : X \to N$).

The fact that f_g does not preserve the monoidal product judgmentally (but rather, we had to resort to a proof by induction employing λ_N and α_N) suggests that

there might be other, more "canonical" candidates for a type enjoying the property of being a free monoid. Consider the 1-HIT:

$$\begin{aligned}\mathsf{FM}(X) ::= e_{\mathsf{FM}} : \mathsf{FM}(X) \;&|\; \iota_{\mathsf{FM}} : X \to \mathsf{FM}(X) \;|\; \otimes_{\mathsf{FM}} : \mathsf{FM}(X) \to \mathsf{FM}(X) \to \mathsf{FM}(X)\\ &|\; \alpha_{\mathsf{FM}} : \Pi\,(a,\, b,\, c : \mathsf{FM}(X))\,.\,(a \otimes_{\mathsf{FM}} b) \otimes_{\mathsf{FM}} c = a \otimes_{\mathsf{FM}} (b \otimes_{\mathsf{FM}} c)\\ &|\; \lambda_{\mathsf{FM}} : \Pi\,(b : \mathsf{FM}(X))\,.\,e_{\mathsf{FM}} \otimes_{\mathsf{FM}} b = b\\ &|\; \rho_{\mathsf{FM}} : \Pi\,(a : \mathsf{FM}(X))\,.\,a \otimes_{\mathsf{FM}} e_{\mathsf{FM}} = a. \end{aligned} \tag{3.3}$$

It is clear that $\mathsf{FM}(X) : \mathsf{Mon}$, as the monoid structure is given entirely by its constructors. Again, given a monoid with carrier N and a function $g : X \to N$, we can define a function $f_g : \mathsf{FM}(X) \to N$ by FM-elimination ($\mathsf{rec}_{\mathsf{FM}}$). Following the scheme given in Section 2.6, we can produce such a function by finding, for N, terms corresponding to the constructors of FM. These can be directly extracted from the monoid structure of N: a term $e'_{\mathsf{FM}} : N$ can be given by e_N, and so on. This works swimmingly also for 1-constructors: for example, the required term $\lambda'_{\mathsf{FM}} : \Pi\,(x : N)\,.\,e_N \otimes_N x = x$ can be comfortably defined as $\lambda'_{\mathsf{FM}} :\equiv \lambda_N$. In applying $\mathsf{rec}_{\mathsf{FM}}$, the only term which is not provided by the monoid structure of N is the one corresponding to the constructor ι_{FM}, i.e. a function $X \to N$, which can be given by g.

The function f_g preserves the unit and the product *judgmentally*: indeed, on 0-constructors, it computes:

$$f_g(e_{\mathsf{FM}}) \equiv e_N \qquad\qquad f_g(a \otimes_{\mathsf{FM}} b) \equiv f_g(a) \otimes_N f_g(b)$$

for every $a, b : \mathsf{FM}(X)$; moreover, $f_g \circ \iota_{\mathsf{FM}} \equiv g$. Conversely, given a monoid homomorphism $h : \mathsf{FM}(X) \to N$, precomposition with ι_{FM} gives a function $h \circ \iota_{\mathsf{FM}} : X \to N$, and it is rather easy to see that $f_{(h \circ \iota_{\mathsf{FM}})}$ and h compute to the same terms on the 0-constructors of FM (for example, $f_{(h \circ \iota_{\mathsf{FM}})} \circ \iota_{\mathsf{FM}} \equiv h \circ \iota_{\mathsf{FM}}$). This suggests a correspondence between monoid homomorphisms $\mathsf{FM}(X) \to N$ and functions $X \to N$: specifically, the kind we normally use to verify adjunctions and to prove freeness of a functor (one which is left adjoint to a forgetful one).

As a monoid, $\mathsf{FM}(X)$ can, indeed, be proved to be free over X. The type $\mathsf{list}(X)$ is too [Uni13, Lemma 6.11.5], but the advantage of considering $\mathsf{FM}(X)$ lies in the fact that the proof of freeness is *contained* in the very definition of $\mathsf{FM}(X)$ – to be precise, in its elimination rule (compare with the discussion on free groups in [Uni13, Chapter 6]). Since verifying that a monoid M is free over X corresponds to showing that it satisfies the universal property defining $\mathsf{FM}(X)$, proving its freeness is equivalent to proving its (monoidal) equivalence to $\mathsf{FM}(X)$. This is, of course, trivial for $\mathsf{FM}(X)$, but less so for $\mathsf{list}(X)$.

The type $\mathsf{list}(X)$, however, is an inductive type with two constructors, whereas $\mathsf{FM}(X)$ is a *higher* inductive type with several 0- and 1-constructors. This makes $\mathsf{list}(X)$, to all intents and purposes, much easier to handle than $\mathsf{FM}(X)$. For example, it is easy to show that $\mathsf{list}(X)$ is a 0-type whenever X is a 0-type (Lemma 3.27); roughly, this is based on the fact that, for every x_i and l_i, there is an equivalence

$$(x_1 :: l_1 = x_2 :: l_2) \simeq (x_1 = x_2) \times (l_1 = l_2)$$

and, by induction, the type on the right-hand side is (-1)-type if X is a 0-type. In contrast, it is not immediately clear that the type

$$a_1 \otimes_{\mathsf{FM}} b_1 = a_2 \otimes_{\mathsf{FM}} b_2$$

is a (-1)-type for every (or any) $a_i, b_i : \mathsf{FM}(X)$, when X is a 0-type.

In short, equivalent constructions respond to different practical needs when it comes to building a proof. HITs are useful because of their ability to package complex universal properties explicitly into their elimination principles, but at the same time they are likely to make other results much harder to reach, in comparison to other constructions of equivalent types. A clear instance of this incongruity emerged in Example 2.133: in light of the fact that a filled triangle is contractible, and hence equivalent to the unit type $\mathbf{1}$, the elimination principle of $\blacktriangleright$ appears to be comically convoluted. Another famous – and far more interesting – example is Sojakova's proof of equivalence between a 2-HIT representing the torus and the product of two circles in HoTT [Soj16], which shows how intrinsically demanding a proof by induction on a (simple) 2-HIT might get, even when pursuing a seemingly easy goal.

Resolving the tension between the different purposes a construction should serve, and hence between different choices for the definition of the same object, is a central issue in this thesis. For example, some of the theorems we formalize will be of the form: "*a mathematical object defined by means of a certain universal property also satisfies a property P.*" The strategy to prove them will be the following:

- defining a HIT C, tailored in such a way that its elimination rule expresses the desired universal property;
- proving $P(C')$ for a type C' with a simpler definition than C;
- establishing an equivalence between C and C', thereby obtaining $P(C)$.

The first case study will be a revisited form of the theorem of coherence for monoidal categories. Selected parts of the formalization of the results of this chapter are featured in Appendix A.1.

3.2 Classical Monoidal Categories

We start by recalling the classical definitions of the mathematical objects and the results we want to formalize in HoTT. The main references for this section are [ML98] and [Lei04].

A **monoidal structure** for a category $\mathcal{C}$ is the data consisting of a bifunctor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, which serves as product, and a chosen object $e \in \mathsf{ob}(\mathcal{C})$, which serves as unit, such that, up to natural isomorphism, the product is associative and unital with respect to the unit object; i.e. there are natural isomorphisms

$$\begin{aligned} \alpha_{a,b,c} : &\quad (a \otimes b) \otimes c \cong a \otimes (b \otimes c) && \text{(associativity)} \\ \lambda_b : &\quad e \otimes b \cong b && \text{(left unitality)} \\ \rho_a : &\quad a \otimes e \cong a && \text{(right unitality)} \end{aligned}$$

which make, moreover, the *coherence diagrams* in Fig. 3.1 commute. We call **monoidal category** a category $\mathcal{C}$ equipped with a monoidal structure, denoting it by $\langle \mathcal{C}, \otimes, e \rangle$. A monoidal structure or category is said to be **strict** – as opposed to *weak* – if the natural isomorphisms α, λ and ρ are identities (in which case, the requirement about the coherence diagrams is trivially satisfied).

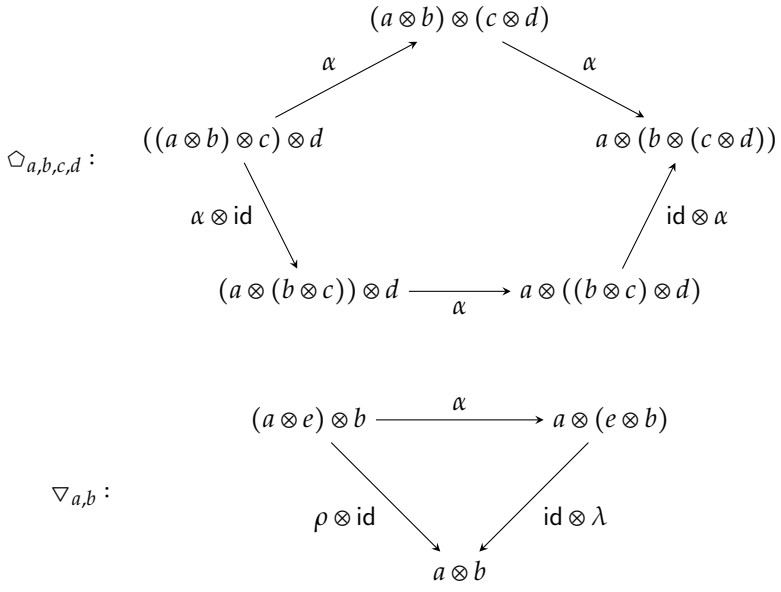


Figure 3.1: The coherence diagrams $\pentagon$ ("Mac Lane's pentagon") and $\triangledown$ in a monoidal category.

A (strong[2]) **monoidal functor** $F : \langle \mathcal{C}, \otimes, e\rangle \to \langle \mathcal{D}, \boxtimes, u\rangle$ between monoidal categories is a functor $F : \mathcal{C} \to \mathcal{D}$ together with isomorphisms in $\mathcal{D}$:

$$F_0 : u \to F(e) \qquad \text{and} \qquad F_2 : F(a) \boxtimes F(b) \to F(a \otimes b)$$

for all $a, b \in \mathsf{ob}(\mathcal{C})$ and natural in both, respecting α, λ and ρ (i.e. making certain diagrams commute; the precise definition will be given for the HoTT implementation in Section 3.3). A **monoidal natural transformation** (resp. isomorphism) between monoidal functors F and G is a natural transformation (resp. isomorphism) that respects the $(-)_0$ and $(-)_2$ components of F and G.

Other notions of monoidal categories exist. For example, "unbiased" monoidal categories are endowed with an n-ary product $\otimes_n : \mathcal{C}^n \to \mathcal{C}$ for every $n \geq 0$ (as opposed to just a "nullary product" $e : \mathcal{C}^0 \to \mathcal{C}$ and a binary product $\otimes$), and dedicated structural isomorphisms, but we do not investigate this notion in this thesis.

Monoidal categories satisfy a theorem of coherence, which has several, equivalent formulations.

Theorem 3.4 (Coherence)**.** *The following (equivalent) statements hold:*

(i) any monoidal category is equivalent, via a strong monoidal functor, to a strict *monoidal category;*

(ii) in a monoidal category, every diagram built out of instances of α, λ and ρ commutes;

(iii) in a free *monoidal category, every diagram commutes.*

Mac Lane's Proof of Coherence

A famous proof of statement (ii) of Theorem 3.4 appears in [ML98, Chapter VII]; we will now briefly summarize it, while omitting several details. A preliminary result of coherence for (representations of) *binary trees* is established. The binary trees we consider have leaves labelled j and z; one can define this structure inductively, declaring that j and z are binary trees, and if t_1 and t_2 are binary trees, so is $t_1 \boxtimes t_2$. Binary trees have a semantics that interprets them as elements in the free monoid generated by j, with z being interpreted to the unit element and $\boxtimes$ to the product in the monoid.

[2]The naming varies among sources; for example, in [Lei04] strong monoidal functors are called *weak*. In general, one distinguishes between *lax*, *colax* and *strong* monoidal functors, depending on the direction of the arrows F_0 and F_2 and whether they are invertible; since we will only consider the latter, we will often leave the designation implicit.

Every binary tree has a j-*length*, counting the occurrences of j: so the trees z and j have j-length 0 and 1 respectively, while the j-length of a tree $t_1 \boxtimes t_2$ is the sum of the j-lengths of t_1 and t_2. For example, $(\mathsf{j} \boxtimes \mathsf{j}) \boxtimes (\mathsf{z} \boxtimes \mathsf{j})$ is a binary tree of j-length 3.

For every $n \in \mathbb{N}$, a directed graph G_n is built, with all the binary trees of j-length n as vertices, and edges defined inductively as follows. There are edges labelled:

$$1_t : t \to t, \qquad \alpha_{t_1,t_2,t_2} : (t_1 \boxtimes t_2) \boxtimes t_3 \to t_1 \boxtimes (t_2 \boxtimes t_3),$$
$$\lambda_t : \mathsf{z} \boxtimes t \to t, \qquad \rho_t : t \boxtimes \mathsf{z} \to t,$$

for every tree t, t_1, t_2 and t_3; moreover, if $f : t_1 \to t_2$ is an edge, then there are edges labelled:

$$f \boxtimes 1_t : t_1 \boxtimes t \to t_2 \boxtimes t \qquad \text{and} \qquad 1_t \boxtimes f : t \boxtimes t_1 \to t \boxtimes t_2,$$

for every tree t. For any n, the graph G_n is closed under these rules.

Every tree t is then assigned a positive integer value describing its *complexity*, i.e., how much it diverges from a "canonical" tree $\widetilde{t}$ of the same length (its *normal form*), of complexity 0 – for example, the one whose representation has all the parentheses lean on one side. For instance, if $t = (\mathsf{j} \boxtimes \mathsf{j}) \boxtimes (\mathsf{j} \boxtimes \mathsf{j})$, then $\widetilde{t} = \mathsf{j} \boxtimes (\mathsf{j} \boxtimes (\mathsf{j} \boxtimes \mathsf{j}))$.

Every path $t_1 \xrightarrow{*} t_2$ in G_n is shown to factor through the normal form, via paths $t_1 \xrightarrow{*} \widetilde{t} \xleftarrow{*} t_2$, which decrease the complexity of the word, in practice showing confluence for the rewriting system given by the edges specified above. Uniqueness of arrows $t \xrightarrow{*} \widetilde{t}$ can be proved "combinatorially", by induction on the complexity of t; we omit the technical details here (in particular, unitality requires a separate argument than associativity).

We then consider the category $\mathcal{T}$ with binary trees as objects and a unique arrow between two binary trees of the same j-length. The category $\mathcal{T}$ is a monoidal category, with z as unit and $\boxtimes$ as monoidal product. Moreover, as a monoidal category, it is free: indeed, given a monoidal category $\langle \mathcal{M}, \otimes, e \rangle$ and an object $a \in \mathcal{M}$, a monoidal functor $\mathcal{T} \to \mathcal{M}$ is defined by sending a tree $t \in \mathcal{T}$ to the monoidal expression in $\mathcal{M}$ obtained by substituting j with a, z with e and $\boxtimes$ with $\otimes$, and any morphism $t_1 \to t_2$ to the unique composition of instances of the associativity and unitality morphisms in $\mathcal{M}$ corresponding to the path obtained by means of the proof above. For any object $a \in \mathcal{M}$, all diagrams in $\mathcal{M}$ built out of instances of α, λ and ρ applied to a have a counterpart in $\mathcal{T}$, and hence they commute.

The argument extends to diagrams involving different objects of $\mathcal{M}$, as follows. Consider the category $\overline{\mathcal{M}}$ of functors $\mathcal{M}^n \to \mathcal{M}$ (for any finite n). This is a monoidal category: the unit is the functor $E : \mathcal{M}^0 \to \mathcal{M}$ sending the point to the unit e of $\mathcal{M}$,

while the product of two functors $F : \mathcal{M}^{n_1} \to \mathcal{M}$ and $G : \mathcal{M}^{n_2} \to \mathcal{M}$ is the functor $F \oplus G : \mathcal{M}^{n_1+n_2} \to \mathcal{M}$, defined by:

$$(F \oplus G)(a_1, \ldots, a_{n_1}, b_1, \ldots, b_{n_2}) \coloneqq F(a_1, \ldots, a_{n_1}) \otimes G(b_1, \ldots, b_{n_2}).$$

Since $\mathcal{T}$ is free, a morphism $\theta : \mathcal{T} \to \overline{\mathcal{M}}$ of monoidal categories is uniquely determined given the image of j, which we take to be the identity functor $\mathsf{id}_{\mathcal{M}} : \mathcal{M} \to \mathcal{M}$. As such, θ sends a binary tree t of j-length n to the functor $\theta(t) : \mathcal{M}^n \to \mathcal{M}$ which substitutes j with the identity functor; that is:

$$\theta(\mathsf{z}) \coloneqq E, \qquad \theta(\mathsf{j}) \coloneqq \mathsf{id}_{\mathcal{M}}, \qquad \theta(t_1 \boxtimes t_2) \coloneqq \theta(t_1) \oplus \theta(t_2). \tag{3.5}$$

Moreover, θ sends morphisms in $\mathcal{T}$ to natural transformations which are built out of instances of associativity and unitality in $\mathcal{M}$. Since all diagrams commute in $\mathcal{T}$, then so do the corresponding diagrams in $\overline{\mathcal{M}}$ between functors, and also those in $\mathcal{M}$ obtained by application of such functors. Fig. 3.2 shows an example of this machinery in motion.

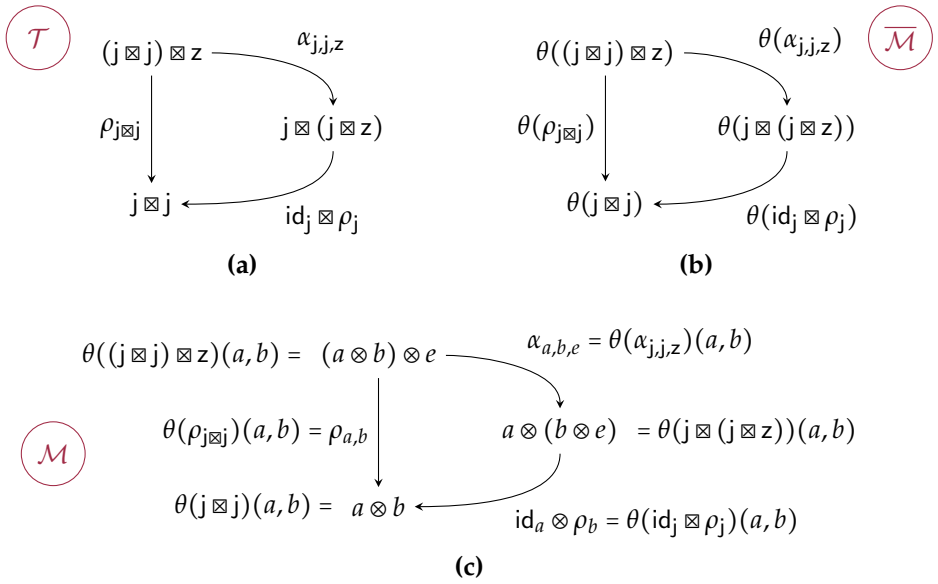


Figure 3.2: An example illustrating Mac Lane's proof of coherence for monoidal categories. The diagram in **(a)** between binary trees of j-length 2 in the category $\mathcal{T}$ commutes by uniqueness of arrows; the functor θ in (3.5) transports such diagram to the one in **(b)** in the category $\overline{\mathcal{M}}$ of functors, which then also commutes. Then, for every $a, b \in \mathcal{M}$, the diagram in **(c)** in the monoidal category $\mathcal{M}$ commutes. Mac Lane's proof shows that any diagram in $\mathcal{M}$ built out of instances of products of arrows α, λ and ρ can be constructed in such a way.

Formalizing Coherence

In formalizing coherence in HoTT, we choose to prove statement (iii) of Theorem 3.4. Our starting point is the proof presented by Beylin and Dybjer [BD96], where coherence is formulated in MLTT and formalized with the proof assistant ALF [Mag95] (using Axiom K, i.e., in HoTT terms, all types are 0-types); the same work was later formalized in [ABD96] with the proof assistant HOL [Gor91].

The informal argument is the following: each monoidal expression in a free monoidal category $\mathcal{F}$ has a normal form in the category, given e.g. by removing all instances of the unit from a product of objects, and by re-associating products so that the parentheses lean on the right; for example:

$$((a \otimes e) \otimes (b \otimes a)) \otimes b \quad \rightsquigarrow \quad a \otimes (b \otimes (a \otimes b)).$$

The discrete subcategory $\mathcal{N}$ of normal forms is a (strict) monoidal category, with a product defined so that source and target of the associativity and unitality morphisms coincide, and hence the morphisms can be defined to be identity arrows. There are strong monoidal functors $J : \mathcal{N} \to \mathcal{F}$ (which is the inclusion) and $K : \mathcal{F} \to \mathcal{N}$ (performing the normalisation); moreover, there is a monoidal natural isomorphism $J \circ K \Rightarrow \mathsf{id}$, the existence of which is essentially a proof that the process of normalisation can be achieved (recursively) via instances of associativity and unitality. Since every diagram commutes in $\mathcal{N}$, so does every diagram in $\mathcal{F}$.

In [BD96], a category was defined as a set of objects together with a family of (hom-)*setoids* for the morphisms; a setoid is a way of simulating a quotient by specifying a set and, separately, an equivalence relation on it. This means that, for every term a, b in the set of objects, a set $\hom(a, b)$ of morphisms is provided and, for every f, g in such a set, another set $E(f, g)$ of equalities between them is defined. A large number of structural equalities ought to appear in the family E. For example, one needs to make sure that E encodes an equivalence relation (so we need terms in $E(f, f)$ for every f; functions $E(f, g) \to E(g, f)$; and so on), but also that the category axioms are satisfied (e.g. a term in $E(f \circ \mathsf{id}_a, f)$). For monoidal categories, the list of equalities to be included in E grows substantially longer; these include: the behaviour of the tensor product and its interplay with the composition in the category; naturality and inverse laws for associativity and unitality of the product; and, of course, the coherence conditions.

This set-up – and the proof of coherence – can obviously be replicated in HoTT entirely. However, working with setoids is notoriously laborious and cumbersome, because every construction depending on a setoid must respect the equivalence

relation; that is, one needs to verify that every function out of a setoid is constant on equivalent terms.[3] For example, if we need to define a functor $F : \mathcal{C} \to \mathcal{D}$ between categories defined as described above, we need to provide a function on hom-sets

$$\overrightarrow{F} : \Pi\,(a, b : \mathsf{ob}(\mathcal{C}))\,.\,\hom_{\mathcal{C}}(a, b) \to \hom_{\mathcal{D}}(F(a), F(b)) \tag{3.6}$$

preserving identity morphisms and composition, and then verify that $\overrightarrow{F}$ sends morphisms in $\mathcal{C}$ in the same equivalence class to morphisms in $\mathcal{D}$ in the same equivalence class. Instead, we can use the built-in equivalence relation on terms given by the identity types. Indeed, in general, a category in HoTT is specified by a type X of objects and, for the morphisms, a family hom containing "identity" terms $1_{(-)}$ and endowed with a composition operation $\circledcirc$, such that the paths in X and the *isomorphisms* in the category are equivalent notions; that is, there is an equivalence

$$(a =_X b) \;\simeq\; \Sigma\,(f : \hom(a, b))\,.\,\Sigma\,(g : \hom(b, a))\,.\,(g \circledcirc f = 1_a) \times (f \circledcirc g = 1_b) \tag{3.7}$$

for every a and $b : X$. In this way, for a functor $F : \mathcal{C} \to \mathcal{D}$ we must still provide a function $\overrightarrow{F}$ as in (3.6), but all we need to check is that $\overrightarrow{F}$ preserves the identity morphisms and the composition.

Coherence for Monoidal Groupoids

We can now apply a further simplification. If we consider a 1-type X and define $\hom(a, b) :\equiv (a =_X b)$, with the identity arrows given by refl and the composition given by the concatenation of paths, the condition in (3.7) is trivially verified. In this way, all the information about objects, morphisms and commutative diagrams in a category is conveniently packed in just *one* type (and its relevant levels of identity types); these correspond, respectively, to terms, paths and 2-paths. Moreover, for a functor F between two such categories, the functorial action $\overrightarrow{F}$ on morphisms is just given by $[F]$, which, as we saw in Section 2.4, does respect reflexivity paths (judgmentally) and concatenation (provably). In other words, we fully draw from the ∞-groupoid structure of a type, which scales down to just a groupoid structure when considering 1-type. The drawback is that, in this way, we are only able to design categories in which all arrows are invertible, i.e., *groupoids*, because every path has an inverse.

Nonetheless, we argue that a result of coherence can still be obtained. Indeed, if we want to prove statement (iii) of Theorem 3.4, we are only interested in *free*

[3] Separately dealing with a set and an equivalence relation defined on it, rather than considering the ensuing quotient, is generally understood to be a problematic approach, paving the road to what is informally known as "*setoid hell*" – a place that HoTT should help escaping [Buc19].

monoidal categories. Since all arrows in a free monoidal category are built up on instances of α, λ and ρ only, they are indeed invertible. Hence a free monoidal category is actually a groupoid, which we can then formalize as a specific 1-type, as we described above. For the reasons explained in Section 3.1, since a free monoidal category over a (0-)type X is defined in terms of its universal property, it is convenient to formalize it as a 1-truncated HIT, which we call $\mathsf{FMG}(X)$. Proving coherence then entails showing that "all diagrams commute" in the ensuing construction; since commutative diagrams in the category are 2-paths in the type, all we need to prove is that $\mathsf{FMG}(X)$ is actually a 0-type.

Coherence for monoidal categories can then be established using a proof by normalisation similar to the one used in [BD96] and described above, provided that we can prove that the construction FMG is free as a monoidal category. Although freeness of a category with respect to monoidal categories follows from freeness with respect to monoidal 1-types, the latter is much easier to formulate and prove. Since we are actually more interested in the proof of normalisation (summarized in Remark 3.53 after the proof of coherence), in how it can be generalized to symmetric monoidal structures (Chapter 4) and in how to exploit HITs to formalize free objects, we will just use 1-types as framework, and declare our result a proof of coherence for monoidal *groupoids*.

Incidentally, working with 1-types leads to a few interesting observations. Strictness of a monoidal structure is a property that cannot be directly expressed in our framework, because it is not homotopy invariant. Indeed, strictness and discreteness are features related to judgmental equalities, over which we do not have control. This is enforced in our theory: a monoidal equivalence of groupoids rests on an equivalence of the corresponding types; if there is such an equivalence, then the types are indistinguishable in the theory, and so there cannot exist a property (e.g. strictness) telling them apart.

In a different direction, the translation from categories to 1-types implies a "relaxation" of certain strict categorical properties: for example, associativity of the composition of the arrows in a category is strict, while associativity of concatenation of paths is only associative up to a coherent choice of higher paths. Hence the result we formalize should more correctly be recognized as "coherence for monoidal *weak* groupoids" (note the order of the adjectives).

We will now proceed to construct free monoidal groupoids in HoTT and prove coherence. We will begin by defining monoidal groupoids, functors and natural

isomorphisms, and by formalizing a list of requirements that a construction should satisfy in order to be considered a free monoidal groupoid.

Remark 3.8 (Notation in figures). As explained, commutativity of categorical diagrams will correspond to the (constructive) existence of specific 2-paths, which are unique up to homotopy: indeed, if G is a 1-type, then by definition all types of paths in G are 0-types, and hence all types of 2-paths in G are (-1)-types. To improve readability, these 2-paths will be presented in form of pictures, with the important caveat that any such picture will strictify the "relaxed" properties mentioned above (e.g. one cannot see which bracketing a concatenation of paths is meant to be read with). With this philosophy in mind, when we will need to prove the commutativity of a diagram, the pictures presented should be thought as already "simplified", where the original diagram can be reconstructed by precomposing with some whiskering corresponding to properties provable by path induction.

In figures, we will use double arrows $\Longrightarrow$ for paths, to keep track of the direction they are defined with, although they always represent invertible arrows; we will use, moreover, triple lines $\Longequal$ for judgmental equalities, and the notation $1 :\equiv \mathsf{refl}$ for the identity path. In large diagrams, we will occasionally replace some bulky terms by the symbol $\bullet$, for them to fit in the figure.

To improve readability, many of the figures we refer to are moved to Section 3.8 at the end of this chapter.

3.3 Monoidal Groupoids

Definition 3.9. A **groupoid** is the data given by a type G and a proof that G is a 1-type. We call Gpd the subuniverse of 1-types, i.e.:

$$\mathsf{Gpd} :\equiv \Sigma\,(G : \mathcal{U})\,.\,\mathsf{IsHGpd}(G).$$

We will generally use the same notation for a groupoid $G : \mathsf{Gpd}$ and for its underlying type $\mathsf{pr}_1(G)$. A **functor** F between groupoids G and G' is simply a function $F : G \to G'$ between the underlying types. A **natural isomorphism** θ between two functors $F, F' : G \to G'$ is a homotopy $\theta : F \sim F'$; this definition is justified by Remark 2.105.

Remark 3.10. Given a groupoid $G : \mathsf{Gpd}$, we do *not* have access to its objects directly, i.e., we cannot extract from G the discrete subcategory of its objects (notice that $\|G\|_0$ gives only the connected components of G, namely, the 0-type of isomorphism classes of the objects in G).

Definition 3.11. A **monoidal structure** on a type M is the data consisting of a unit term in M, a product $M \to M \to M$ and families of paths and 2-paths witnessing 1-coherent associativity and unitality of the product with respect to the unit. Precisely, given a type M, we define the type $\mathsf{MonStructure}(M)$ as the Σ-type encoding the following data, which we present as a bulleted list:

- $e_M : M$ (unit);
- $\otimes_M : M \to M \to M$ (monoidal product);
- $\alpha_M : \Pi\,(a, b, c : M)\,.\,(a \otimes_M b) \otimes_M c = a \otimes_M (b \otimes_M c)$ (associativity);
- $\lambda_M : \Pi\,(b : M)\,.\,e_M \otimes_M b = b$ (left unitality);
- $\rho_M : \Pi\,(a : M)\,.\,a \otimes_M e_M = a$ (right unitality);
- families $\pentagon_M$ and $\triangledown_M$ of 2-paths filling the coherence diagrams in Fig. 3.1, with instances of α_M, λ_M and ρ_M *in lieu* of the arrows, i.e.:

$$\begin{aligned}\pentagon_M : \Pi\,(a, b, c, d : M)\,.\,&\alpha_M(a \otimes_M b, c, d) \cdot \alpha_M(a, b, c \otimes_M d) \\ &= (\alpha_M(a, b, c) \otimes_M \mathsf{refl}_d) \cdot \alpha_M(a, b \otimes_M c, d) \cdot (\mathsf{refl}_a \otimes_M \alpha_M(b, c, d)), \qquad (3.12)\end{aligned}$$

$$\triangledown_M : \Pi\,(a, b : M)\,.\,\alpha_M(a, e_M, b) \cdot (\mathsf{refl}_a \otimes_M \lambda_M(b)) = \rho_M(a) \otimes_M \mathsf{refl}_b, \qquad (3.13)$$

where we used the notation as in Remark 2.65 for product of paths. We will sometimes omit the arguments of α_M, λ_M, ρ_M, $\pentagon_M$ and $\triangledown_M$ if they can be inferred from the context. The type of **monoidal groupoids** is then defined as:

$$\mathsf{MonGpd} :\equiv \Sigma\,(M : \mathsf{Gpd})\,.\,\mathsf{MonStructure}(M);$$

we will use the same notation for a monoidal groupoid $M : \mathsf{MonGpd}$ and its *carrier* $\mathsf{pr}_1(\mathsf{pr}_1(M))$.

Remark 3.14. In general, the interchange law in (2.64) interacts with the one in (2.49) and leads to the validity of "higher interchange laws", defined as specific higher paths; however, since $\otimes_M$ is defined for 1-types, all 3-paths and higher are already filled, so we will not need to specify them. Indeed, the given definition of monoidal structure on a type does not take into account higher coherence diagrams, as in this thesis it is meant to be applied to groupoids only, as opposed to ∞-groupoids; in general, a term $s : \mathsf{MonStructure}(M)$ witnesses a 1-*coherent* monoidal structure for the type M (borrowing the terminology from [Bru16]).

Classically, the associativity and unitality isomorphisms are required to be *natural* in all their arguments. In our framework, naturality of associativity and unitality is expressed in terms of the application of $\otimes_M$ to paths, and hence it can be proved by path induction.

Lemma 3.15. *Associativity and unitality in a monoidal structure are natural in all their arguments, i.e., given paths $p : a =_M a'$, $q : b =_M b'$ and $r : c =_M c'$, there are 2-paths filling the squares in Fig. 3.3.*

Proof. By induction on p, q and r, the squares are trivially filled. □

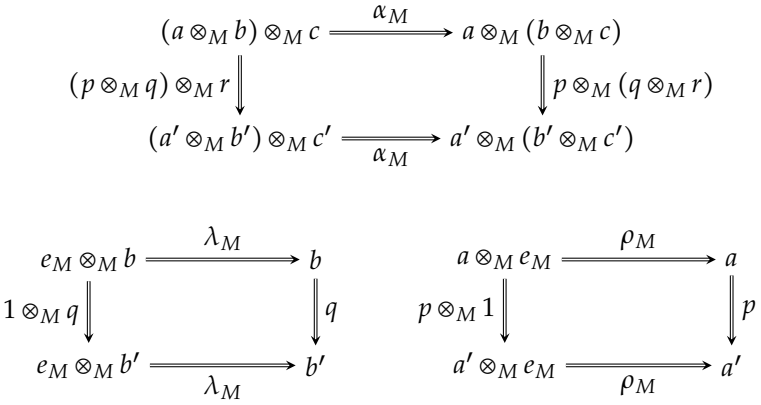


Figure 3.3: Naturality of α_M, λ_M and ρ_M.

Naturality of associativity and unitality, together with $\pentagon_T$ and $\triangledown_T$, combine into other noteworthy diagrams [Kel64].

Lemma 3.16. *The additional coherence diagrams in Fig. 3.4 commute for every $a, b : M$.*

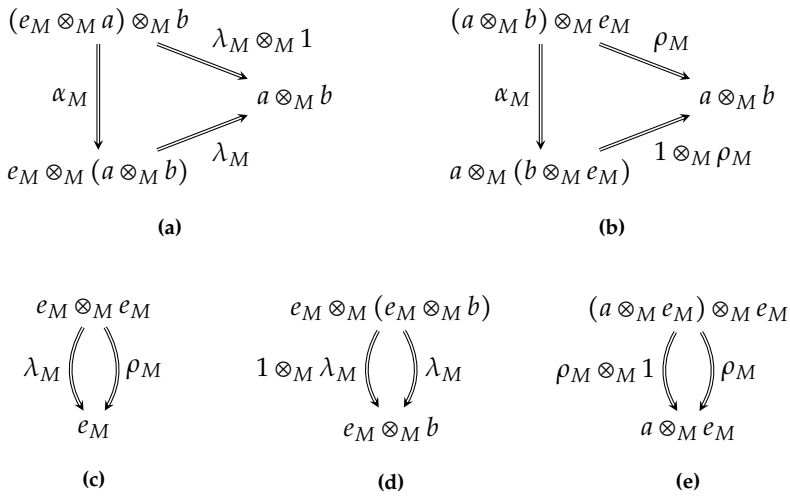


Figure 3.4: Additional coherence diagrams in a monoidal category.

Proof. The construction of the 2-paths in Fig. 3.4a and Fig. 3.4c is shown in Fig. 3.9a and Fig. 3.9b respectively; commutativity of the diagram in Fig. 3.4b is proved similarly as for Fig. 3.4a. The diagram in Fig. 3.4d does not involve the coherence diagrams $\pentagon_M$ and $\triangledown_M$; rather, it is derived from the naturality square for λ_M (Lemma 3.15), with $q :\equiv \lambda_M(b)$. The diagram in Fig. 3.4e is similarly derived from the naturality square for ρ_M. □

Definition 3.17. The type of (strong) **monoidal functors** between two monoidal groupoids $\langle M, e_M, \otimes_M, \ldots\rangle$ and $\langle N, e_N, \otimes_N, \ldots\rangle$ is defined as the Σ-type encoding the following data:

- a functor $F : M \to N$, i.e., a function between the underlying types;
- a path $F_0 : e_N = F(e_M)$;
- a family of paths $F_2 : \Pi\,(a, b : M)\,.\,F(a) \otimes_N F(b) = F(a \otimes_M b)$;
- families F_α, F_λ and F_ρ of 2-paths corresponding to the diagrams in Fig. 3.5, for every $a, b, c : M$.

We will denote by $\mathsf{MonGpd}(M, N)$ the resulting type $\Sigma\,(F : M \to N)\,.\,\Sigma\,(F_0 : \ldots)\,.\ldots$ and use the same notation for a monoidal functor $F : \mathsf{MonGpd}(M, N)$ and its underlying function $\mathsf{pr}_1(F)$.

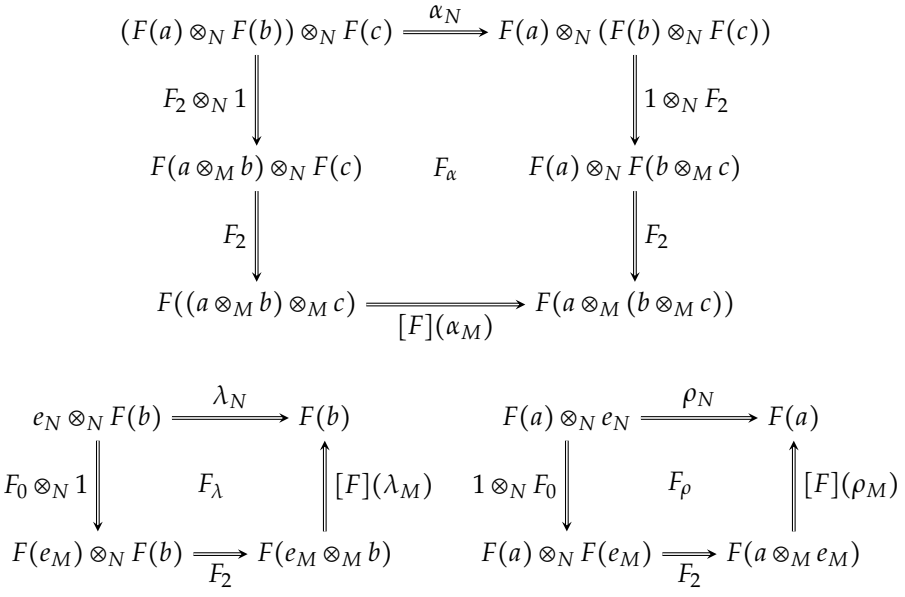


Figure 3.5: Coherence conditions for monoidal functors.

Our implementation of monoidal functors is consistent with the classical definition. Given any $M : \mathsf{MonGpd}$, we can construct an identity monoidal functor $\mathsf{id}_M : \mathsf{MonGpd}(M, M)$ with the identity map $\mathsf{id}_M : M \to M$ as underlying function. Moreover, we can define a composition $(G \circ F) : \mathsf{MonGpd}(M, P)$ of two monoidal functors $F : \mathsf{MonGpd}(M, N)$ and $G : \mathsf{MonGpd}(N, P)$, in the following way:

- the function underlying the composition $G \circ F$ is the composition of the underlying functions G and F;
- the path $(G \circ F)_0 : e_P = G(F(e_M))$ is defined as $(G \circ F)_0 :\equiv G_0 \cdot [G](F_0)$;
- for all $a, b : M$, the path $(G \circ F)_2(a, b) : G(F(a)) \otimes_P G(F(b)) = G(F(a \otimes_M b))$ is given by $(G \circ F)_2(a, b) :\equiv G_2 \cdot [G](F_2(a, b))$;
- $(G \circ F)_\alpha$, $(G \circ F)_\lambda$ and $(G \circ F)_\rho$ are the 2-paths displayed in Figs. 3.10a and 3.10b.

Definition 3.18. The type of **monoidal natural isomorphisms** between monoidal functors F and $G : \mathsf{MonGpd}(M, N)$ is defined as the Σ-type encoding:

- a natural isomorphism (i.e., a homotopy) $\theta : F \sim G$ between the underlying functors;
- a 2-path θ_0 and a family θ_2 of 2-paths, corresponding to the diagrams in Fig. 3.6, for every $a, b : M$.

We will denote by $\mathsf{MonFun}_{M,N}(F, G)$ the resulting Σ-type and use the same notation for a monoidal natural isomorphism $\theta : \mathsf{MonFun}_{M,N}(F, G)$ and its underlying natural isomorphism.

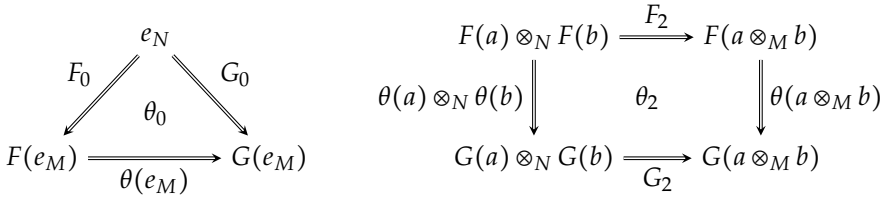


Figure 3.6: Coherence conditions for a monoidal natural isomorphism.

Monoidal natural isomorphisms are, of course, invertible, and can be composed vertically (using as underlying homotopy the composition in Definition 2.83) or horizontally, although the latter will not be used in this thesis. Observe that no conditions concerning the $-_\alpha$, $-_\lambda$ and $-_\rho$ components of F and G are required in the definition of a monoidal natural isomorphism (indeed they would correspond to 3-paths in N, which are trivially given by the proof that N is a 1-type).

Given $F : \mathsf{MonGpd}(M,N)$, $G : \mathsf{MonGpd}(N,P)$ and $H : \mathsf{MonGpd}(P,Q)$, one can construct a monoidal natural isomorphism

$$\theta : \mathsf{MonFun}_{M,Q}((H \circ G) \circ F, H \circ (G \circ F)),$$

i.e., that the composition of symmetric monoidal functors is associative. Explicitly:

- the underlying natural isomorphism θ is given pointwise by the identity path, since composition of functions is judgmentally associative;
- θ_0 and θ_2 are proved by the fact that the application of functions to paths respects composition of functions and concatenation of paths (note that the monoidal product $\otimes$ preserves identity paths).

Similarly, one can show that the identity monoidal functor is a left and right unit for the composition of monoidal functors.

Definition 3.19. Given $F : \mathsf{MonGpd}(M,N)$ and $G : \mathsf{MonGpd}(N,M)$, the underlying homotopies in any two monoidal natural isomorphisms

$$\epsilon : \mathsf{MonFun}_{M,M}(G \circ F, \mathsf{id}_M) \qquad \text{and} \qquad \eta : \mathsf{MonFun}_{N,N}(\mathsf{id}_N, F \circ G)$$

prove that the underlying functions in F and G are half-adjoint in an equivalence between the carriers of M and N. The quadruple $\langle F, G, \epsilon, \eta \rangle$ will be called a **monoidal equivalence**; the type of monoidal equivalences will be denoted by $M \simeq N$.

Conversely, an equivalence of the underlying types implies a monoidal equivalence in a canonical way.

Lemma 3.20. *If the underlying function of a monoidal functor $F : \mathsf{MonGpd}(M,N)$ is an equivalence of types, then F is a monoidal equivalence; i.e., the inverse $G : N \to M$ of the underlying functor can be promoted to a monoidal functor $G : \mathsf{MonGpd}(N,M)$ which, together with F, realizes a monoidal equivalence $M \simeq N$.*

Proof. If $h : G \circ F \sim \mathsf{id}_M$ and $k : F \circ G \sim \mathsf{id}_N$ are the data of the equivalence, the monoidal components of G are given as follows:

- a path $G_0 : e_M = G(e_N)$ is given by $G_0 :\equiv ([G](F_0) \cdot h(e_M))^{-1}$;
- for all $a, b : N$, a path $G_2(a,b) : G(a) \otimes_M G(b) = G(a \otimes_N b)$ can be constructed:

$$G_2(a,b) :\equiv h(G(a) \otimes_M G(b))^{-1} \cdot [G](F_2(G(a), G(b)))^{-1} \cdot [G](k(a) \otimes_N k(b));$$

- the families G_α, G_λ and G_ρ are similarly obtained from F_α, F_λ and F_ρ; a 2-path for G_α is shown in Fig. 3.11, while G_λ is displayed in Fig. 3.12 and G_ρ is obtained similarly.

The required natural isomorphisms (i.e., homotopies) are provided directly by h and k, while the 2-paths h_0 and k_0 and the families h_2 and k_2 of 2-paths are then found by easy path algebra. □

At this point, $\mathcal{U}$, Gpd and MonGpd all have category-like features: the objects are, respectively, types, 1-types and monoidal groupoids, and each of them comes with an appropriate notion of morphisms, identity and composition. The subuniverse $\mathsf{Set} :\equiv \Sigma\,(S : \mathcal{U})\,.\,\mathsf{IsHSet}(S)$ also does, with 0-types as objects and functions as morphisms. We might ask whether a functorial construction $F : \mathsf{Set} \to \mathsf{MonGpd}$ is *free*. In terms of a hom-set adjunction, from the perspective of category theory, we would like to show that there is an equivalence

$$\mathsf{MonGpd}(F(X), M) \simeq \mathsf{Set}(X, \mathsf{ob}(M)) \tag{3.21}$$

natural in X and M, between the hom-sets of the category of monoidal groupoid and monoidal functors and the category of sets and functions of sets. However, as pointed out in Remark 3.10, we do not have access to the 0-type of objects of a (monoidal) groupoid; hence, a forgetful functor $\mathsf{ob} : \mathsf{MonGpd} \to \mathsf{Set}$ cannot be constructed. However, meta-theoretically, we have a natural isomorphism

$$\mathsf{Set}(X, \mathsf{ob}(M)) \simeq \mathcal{S}(X^\delta, M) \tag{3.22}$$

where $\mathcal{S}$ is the category of spaces and functions of spaces, corresponding to $\mathcal{U}$ in our theory, and $X^\delta :\equiv \mathsf{pr}_1(X)$ is a *discrete* space (a 0-type, ignoring its property of being so). In practice, this means that, if we can verify that a functorial construction $F : \mathcal{U} \to \mathsf{MonGpd}$ is free on types, then the same construction restricted to Set can be considered as free on 0-types.

The notions of functors $F : \mathcal{U} \to \mathsf{MonGpd}$ and of free functors are made precise in the following definitions. These could be adapted to other types or subuniverses with a category-like structure; we will refrain from making these definitions more general, as they would require an internal theory of higher categories that we will not need beyond what is included in this thesis (we stress that the "hom-types" of $\mathcal{U}$ and MonGpd are not sets).

Definition 3.23. A **functor** from $\mathcal{U}$ to MonGpd consists of a function $F : \mathcal{U} \to \mathsf{MonGpd}$, together with a function between "hom-types" (function types)

$$\overrightarrow{F} : \Pi\,(X, Y : \mathcal{U})\,.\,(X \to Y) \to \mathsf{MonGpd}(F(X), F(Y))$$

respecting identity and composition, i.e., with terms

$$F_{\mathsf{id}} : \mathsf{MonFun}_{F(X),F(X)}\left(\overrightarrow{F}(\mathsf{id}_X), \mathsf{id}_{F(X)}\right),$$
$$F_{\circ} : \Pi\,(f : X \to Y,\ g : Y \to Z)\,.\,\mathsf{MonFun}_{F(X),F(Z)}\left(\overrightarrow{F}(g) \circ \overrightarrow{F}(f), \overrightarrow{F}(g \circ f)\right),$$

for every $X, Y, Z : \mathcal{U}$. Without the need for a name for the type of such functors, we will just refer to a function $F : \mathcal{U} \to \mathsf{MonGpd}$ as a "functor" if the remaining data is implied.

Definition 3.24. Let $F : \mathcal{U} \to \mathsf{MonGpd}$ be a functor in the sense of Definition 3.23. We say that F is **free** if it is left adjoint to the forgetful functor to $\mathcal{U}$; i.e., if there are:

- a function $\phi : \Pi\,(X : \mathcal{U})\,.\,\Pi\,(M : \mathsf{MonGpd})\,.\,\mathsf{MonGpd}(F(X), M) \to X \to M$ natural in M, i.e. the diagram in Fig. 3.7a commutes for every $H : \mathsf{MonGpd}(M, N)$;
- a function $\psi : \Pi\,(X : \mathcal{U})\,.\,\Pi\,(M : \mathsf{MonGpd})\,.\,(X \to M) \to \mathsf{MonGpd}(F(X), M)$, natural in X, i.e. the diagram in Fig. 3.7b commutes for every $h : Y \to X$;
- a family of homotopies $\theta : \Pi\,(X : \mathcal{U})\,.\,\Pi\,(M : \mathsf{MonGpd})\,.\,\phi_{X,M} \circ \psi_{X,M} \sim \mathsf{id}_{(X \to M)}$;
- a family of monoidal natural isomorphisms

$$\chi : \Pi\,(X : \mathcal{U})\,.\,\Pi\,(M : \mathsf{MonGpd})\,.\,\mathsf{MonFun}_{F(X),M}(\psi_{X,M}(\phi_{X,M}(G)), G)$$

  for every $G : \mathsf{MonGpd}(F(X), M)$.

If $X : \mathcal{U}$, the monoidal groupoid $F(X)$ is said to be **freely generated** by X.

Remark 3.25. The types of homotopies and of monoidal natural isomorphisms presented in Definition 3.24 describing naturality of ϕ and ψ and the unit and counit of the adjunction (θ and χ) are 0-types; hence, if such terms exist, they might not be unique, accounting for the same functor being free in distinct ways. In contrast, one could choose to require terms in the (-1)-truncation of those types – or even to define the property of being free as the (-1)-truncation of the Σ-type describing ϕ, ψ and their attributes. This choice, which is probably relevant while building an internal theory of higher categories, is not so in our setting, hence we will use the non-truncated definition for simplicity.

As mentioned above, in this thesis we will focus our attention to free monoidal groupoids generated by *sets*. Statement (iii) in Theorem 3.4 then corresponds in our framework to the following.

Theorem 3.26 (Coherence for monoidal groupoids)**.** *A functor $F : \mathcal{U} \to \mathsf{MonGpd}$ exists such that it is free and, for every 0-type X, the carrier of $F(X)$ is a 0-type.*

$$\begin{array}{ccc} \mathsf{MonGpd}(F(X), M) & \xrightarrow{\phi_{X,M}} & (X \to M) \\ {\scriptstyle H \circ -}\downarrow & & \downarrow{\scriptstyle H \circ -} \\ \mathsf{MonGpd}(F(X), N) & \xrightarrow[\phi_{X,N}]{} & (X \to N) \end{array}$$

(a) Naturality of ϕ in M: the diagram commutes for every $H : \mathsf{MonGpd}(M, N)$, i.e. there is a homotopy $H \circ \phi_{X,M}(G) \sim \phi_{X,N}(H \circ G)$ for every $G : \mathsf{MonGpd}(F(X), M)$.

$$\begin{array}{ccc} (X \to M) & \xrightarrow{\psi_{X,M}} & \mathsf{MonGpd}(F(X), M) \\ {\scriptstyle - \circ h}\downarrow & & \downarrow{\scriptstyle - \circ F(h)} \\ (Y \to M) & \xrightarrow[\psi_{Y,M}]{} & \mathsf{MonGpd}(F(Y), M) \end{array}$$

(b) Naturality of ψ in X: the diagram commutes for every $h : Y \to X$, i.e. there is a monoidal natural isomorphism in $\mathsf{MonFun}_{F(Y),M}(\psi_{X,M}(g) \circ F(h), \psi_{Y,M}(g \circ h))$ for every $g : X \to M$.

Figure 3.7: Naturality conditions for ϕ and ψ in the definition of a free functor.

We will achieve this result in several steps. First, we will examine two functors: one (list) easily proved to produce monoidal groupoids that are 0-types; one easily proved to be free. Then we will show that these two functors produce equivalent types.

3.4 Lists as Monoidal Groupoids

A candidate for the free monoidal groupoid generated by a 0-type X is the type $\mathsf{list}(X)$. Before discussing its freeness, we need to make sure that $\mathsf{list}(X)$ is a 1-type. The following lemma will show that it is, in fact, a 0-type.

Lemma 3.27. *Let $X : \mathcal{U}$ and assume that X is an 0-type. Then $\mathsf{list}(X)$ is also a 0-type.*

Proof. We will make use of an "encode-decode" argument to characterize the identity types in $\mathsf{list}(X)$; this technique, presented e.g. in [LS13], systematizes the use of path induction in the proof that a family of paths is equivalent to another family of types.

An encode-decode proof works in the following way. In general, for a type T, one defines a a comparison family $\mathsf{code} : T \to T \to \mathcal{U}$ where, for every t_1, $t_2 : T$, the type $\mathsf{code}(t_1, t_2)$ is a guess for what the type $(t_1 = t_2)$ should be equivalent to. In particular, one needs to find, in the family code, terms corresponding to identity

paths refl_t for every $t : T$, i.e., a function $r : \Pi\,(t : T)\,.\,\mathsf{code}(t,t)$ needs to be defined. A family of equivalences $\Pi\,(t_1, t_2 : T)\,.\,(t_1 = t_2) \simeq \mathsf{code}(t_1, t_2)$ is then constructed; the families of half-adjoint functions take the name of encode (from the identity types to code) and decode. The family $\mathsf{encode} : \Pi\,(t_1, t_2 : T)\,.\,(t_1 = t_2) \to \mathsf{code}(t_1, t_2)$ is defined, for every $p : t_1 = t_2$, by transporting $r(t_1)$ along p in the family $\mathsf{code}(t_1, -)$, i.e.,

$$\mathsf{encode}(t_1, t_2, p) :\equiv p_*^{(t \mapsto \mathsf{code}(t_1, t))}(r(t_1)); \tag{3.28}$$

in particular, $\mathsf{encode}(t, t, \mathsf{refl}_t) \equiv r(t)$ for every $t : T$. The construction of the family $\mathsf{decode} : \Pi\,(t_1, t_2 : T)\,.\,\mathsf{code}(t_1, t_2) \to (t_1 = t_2)$ is problem-specific, but it should be designed in such a way that

$$\mathsf{decode}(t, t, r(t)) = \mathsf{refl}_t \tag{3.29}$$

for every $t : T$. In proving that $\mathsf{encode}(t_1, t_2)$ and $\mathsf{decode}(t_1, t_2)$ are half-adjoint equivalences, one of the steps is then automatic: we have that

$$\Pi\,(t_1, t_2 : T)\,.\,\Pi\,(p : t_1 = t_2)\,.\,\mathsf{decode}(t_1, t_2, \mathsf{encode}(t_1, t_2, p)) = p$$

by induction on p and by (3.29), so $\mathsf{decode}(t_1, t_2)$ is a section of $\mathsf{encode}(t_1, t_2)$. The other direction, again, depends on the specific T and definition of code.

We will use the machinery we just described to prove that $\mathsf{list}(X)$ is a 0-type whenever X is a 0-type; we will do this by showing that the types of paths in $\mathsf{list}(X)$ are equivalent to a family of (-1)-types. We refer to Definition 2.29 for the notation relevant to lists, and to Definition 2.8 for the notation relevant to terms in product types.

Given $l_1, l_2 : \mathsf{list}(X)$, we define the type $\mathsf{code}(l_1, l_2)$ by (double) induction on lists. We declare:

$$\begin{aligned} \mathsf{code}(\mathsf{nil}, \mathsf{nil}) &:\equiv \mathbf{1}, & \mathsf{code}(\mathsf{nil}, x_2 :: l_2) &:\equiv \mathbf{0}, \\ \mathsf{code}(x_1 :: l_1, \mathsf{nil}) &:\equiv \mathbf{0}, & \mathsf{code}(x_1 :: l_1, x_2 :: l_2) &:\equiv (x_1 = x_2) \times \mathsf{code}(l_1, l_2), \end{aligned}$$

for every $x_1, x_2 : X$. Induction on l_1 and l_2 shows that $\mathsf{code}(l_1, l_2)$ is a (-1)-type, as it is either $\mathbf{1}$, $\mathbf{0}$, or the product of an identity type between terms in a 0-type and what is, by the induction hypothesis, a (-1)-type (see Remark 2.75), so we have a term

$$\mathsf{code{-}trunc} : \Pi\,(l_1, l_2 : \mathsf{list}(X))\,.\,\Pi\,(a, b : \mathsf{code}(l_1, l_2))\,.\,a = b.$$

A function $r : \Pi\,(l : \mathsf{list}(X))\,.\,\mathsf{code}(l, l)$ is defined by induction on l, as:

$$\begin{aligned} r(\mathsf{nil}) &:\equiv * : \mathbf{1} \equiv \mathsf{code}(\mathsf{nil}, \mathsf{nil}), \\ r(x :: l) &:\equiv \langle \mathsf{refl}_x, r(l) \rangle : (x = x) \times \mathsf{code}(l, l) \equiv \mathsf{code}(x :: l, x :: l), \end{aligned}$$

for every $x : X$ and $l : \mathsf{list}(X)$. Then a function $\mathsf{encode} : \prod(l_1, l_2 : \mathsf{list}(X)) . (l_1 = l_2) \to \mathsf{code}(l_1, l_2)$ can defined as in (3.28), i.e.,

$$\mathsf{encode}(l_1, l_2, p) :\equiv p_*^{(k \mapsto \mathsf{code}(l_1,k))}(r(l_1)).$$

The identity

$$\mathsf{encode}(x_1 :: l_1, x_2 :: l_2, [\mathsf{cons}](p, q)) =_{\mathsf{code}(x_1 :: l_1, x_2 :: l_2)} \langle p, \mathsf{encode}(l_1, l_2, q)\rangle \quad (3.30)$$

holds for every $p : x_1 = x_2$ and $q : l_1 = l_2$, as proved by induction on p and q, since

$$\begin{aligned}\mathsf{encode}(x_1 :: l_1, x_1 :: l_1, [\mathsf{cons}](\mathsf{refl}_{x_1}, \mathsf{refl}_{l_1})) &\equiv \mathsf{encode}(x_1 :: l_1, x_1 :: l_1, \mathsf{refl}_{x_1 :: l_1}) \\ &\equiv r(x_1 :: l_1) \equiv \langle \mathsf{refl}_{x_1}, r(l_1)\rangle \\ &\equiv \langle \mathsf{refl}_{x_1}, \mathsf{encode}(l_1, l_1, \mathsf{refl}_{l_1})\rangle.\end{aligned}$$

In the other direction, a function $\mathsf{decode} : \prod(l_1, l_2 : \mathsf{list}(X)) . \mathsf{code}(l_1, l_2) \to (l_1 = l_2)$ is defined by induction:

$$\begin{aligned}\mathsf{decode}(\mathsf{nil}, \mathsf{nil}, c) &:\equiv \mathsf{refl}_{\mathsf{nil}}, \\ \mathsf{decode}(x_1 :: l_1, \mathsf{nil}, c) &:\equiv \mathsf{ind}_{\mathbf{0}}(c), \\ \mathsf{decode}(\mathsf{nil}, x_2 :: l_2, c) &:\equiv \mathsf{ind}_{\mathbf{0}}(c), \\ \mathsf{decode}(x_1 :: l_1, x_2 :: l_2, \langle p, q\rangle) &:\equiv [\mathsf{cons}](p, \mathsf{decode}(l_1, l_2, q))\end{aligned}$$

for every $x_1, x_2 : X$ and $l_1, l_2 : \mathsf{list}(X)$. We can now prove that encode and decode are half-adjoint in a family of equivalences. Given $l_1, l_2 : \mathsf{list}(X)$, we have

$$\mathsf{dec{-}enc} : \prod(l_1, l_2 : \mathsf{list}(X)) . \prod(p : l_1 = l_2) . \mathsf{decode}(l_1, l_2, \mathsf{encode}(l_1, l_2, p)) = p,$$

as proved by induction on p and then on l_1, since we can define terms:

$$\mathsf{dec{-}enc}(\mathsf{nil}, \mathsf{nil}, \mathsf{refl}_{\mathsf{nil}}) :\equiv \mathsf{refl}_{\mathsf{refl}_{\mathsf{nil}}}$$

because $\mathsf{decode}(\mathsf{nil}, \mathsf{nil}, \mathsf{encode}(\mathsf{nil}, \mathsf{nil}, \mathsf{refl}_{\mathsf{nil}})) \equiv \mathsf{refl}_{\mathsf{nil}}$, and

$$\mathsf{dec{-}enc}(x_1 :: l_1, x_1 :: l_1, \mathsf{refl}_{x_1 :: l_1}) :\equiv [[x_1 :: -]](\mathsf{dec{-}enc}(l_1, l_1, \mathsf{refl}_{l_1}))$$

for every $x_1 : X$, $l_1 : \mathsf{list}(X)$, because

$$\begin{aligned}\mathsf{decode}(x_1 :: l_1, x_1 :: l_1, \langle \mathsf{refl}_{x_1}, r(l_1)\rangle) &\equiv [\mathsf{cons}](\mathsf{refl}_{x_1}, \mathsf{decode}(l_1, l_1, r(l_1))) \\ &\equiv [x_1 :: -](\mathsf{decode}(l_1, l_1, r(l_1))).\end{aligned}$$

Conversely, we have

$$\mathsf{enc{-}dec} : \prod(l_1, l_2 : \mathsf{list}(X)) . \prod(c : \mathsf{code}(l_1, l_2)) . \mathsf{encode}(l_1, l_2, \mathsf{decode}(l_1, l_2, c)) = c,$$

as proved by induction on l_1 and l_2, by defining:

$$\begin{aligned}
\mathsf{enc{-}dec}(\mathsf{nil}, \mathsf{nil}, c) &:\equiv \mathsf{ind}_{\mathbf{1}}(c, \mathsf{refl}_*) &&: * = c,\\
\mathsf{enc{-}dec}(x_1 :: l_1, \mathsf{nil}, c) &:\equiv \mathsf{ind}_{\mathbf{0}}(c) &&: \mathsf{encode}(x_1 :: l_1, \mathsf{nil}, \mathsf{ind}_{\mathbf{0}}(c)) = c,\\
\mathsf{enc{-}dec}(\mathsf{nil}, x_2 :: l_2, c) &:\equiv \mathsf{ind}_{\mathbf{0}}(c) &&: \mathsf{encode}(\mathsf{nil}, x_2 :: l_2, \mathsf{ind}_{\mathbf{0}}(c)) = c,
\end{aligned}$$

while a term $\mathsf{enc{-}dec}(x_1 :: l_1, x_2 :: l_2, c)$ can be found by induction on the product type of c, as the concatenation:

$$\begin{aligned}
&\mathsf{encode}(x_1 :: l_1, x_2 :: l_2, \mathsf{decode}(x_1 :: l_1, x_2 :: l_2, \langle p, c' \rangle))\\
&\equiv \mathsf{encode}(x_1 :: l_1, x_2 :: l_2, [\mathsf{cons}](p, \mathsf{decode}(l_1, l_2, c')))\\
&= \langle p, \mathsf{encode}(l_1, l_2, \mathsf{decode}(l_1, l_2, c')) \rangle && \text{by (3.30)}\\
&= \langle p, c' \rangle && \text{by } \mathsf{enc{-}dec}(l_1, l_2, c').
\end{aligned}$$

Hence encode and decode produce half-adjoint functions in a family of equivalences, i.e., for every $l_1, l_2 : \mathsf{list}(X)$, we have: $(l_1 = l_2) \simeq \mathsf{code}(l_1, l_2)$.

We are now ready to prove that $\mathsf{list}(X)$ is a 0-type. Given $l_1, l_2 : \mathsf{list}(X)$ and $p, q : l_1 = l_2$, we have a term:

$$t : \mathsf{encode}(l_1, l_2, p) =_{\mathsf{code}(l_1, l_2)} \mathsf{encode}(l_1, l_2, q)$$

given by $t :\equiv \mathsf{code{-}trunc}(l_1, l_2, \mathsf{encode}(l_1, l_2, p), \mathsf{encode}(l_1, l_2, q))$, and hence a 2-path $p = q$ by Remark 2.105 (where the homotopy used is $\mathsf{dec{-}enc}(l_1, l_2)$). □

Lemma 3.31. *If X is a 0-type, then $\mathsf{list}(X)$ has the structure of a monoidal groupoid, defined in the proof.*

Proof. Since X is a 0-type, then $\mathsf{list}(X)$ is a 0-type by Lemma 3.27, and in particular a 1-type, so it is the carrier type in a term $\mathsf{list}(X) : \mathsf{Gpd}$. The operation ++ serves a monoidal product, with nil as unit; this satisfies associativity and unitality as proved in Lemma 3.1 at the beginning of this chapter. Moreover, there are families of 2-paths $\pentagon_{\mathsf{list}}$ and $\triangledown_{\mathsf{list}}$ corresponding to the coherence diagrams, since $\mathsf{list}(X)$ is a 0-type. □

Remark 3.32. The type $\mathsf{list}(X)$ can be given the structure of a monoidal groupoid also if X is a 1-type, by suitably modifying the proof given for Lemma 3.27. The proof of Lemma 3.31 still holds, except that the coherence diagrams are not provided by the truncation level of $\mathsf{list}(X)$. However, a term $\pentagon_{\mathsf{list}}(l_1, l_2, l_3, l_4)$ corresponding to the diagram in Fig. 3.13a can be produced for every $l_i : \mathsf{list}(X)$ by

induction on l_1 (Figs. 3.13b and 3.13c), using the fact that, for all paths p and q in $\mathsf{list}(X)$ and $x : X$, the following 2-paths can be found by path induction:

$$\mathsf{refl}_{\mathsf{nil}} \mathbin{+\!\!+} p = p, \tag{3.33}$$

$$[x :: -](p) \mathbin{+\!\!+} q = [x :: -](p \mathbin{+\!\!+} q). \tag{3.34}$$

A term $\triangledown_{\mathsf{list}}(l_1, l_2)$ can be produced in the same way.

We will use the notation $\mathsf{list} : \mathcal{U} \to \mathsf{MonGpd}$ for the function

$$\mathsf{list} :\equiv \left(X \mapsto \langle \|\mathsf{list}(X)\|_1, \mathsf{nil}, \mathbin{+\!\!+}, \alpha_{\mathsf{list}}, \lambda_{\mathsf{list}}, \rho_{\mathsf{list}}, \pentagon_{\mathsf{list}}, \triangledown_{\mathsf{list}} \rangle\right). \tag{3.35}$$

If X is a 0-type, then $\|\mathsf{list}(X)\|_1 \simeq \mathsf{list}(X)$, as the n-truncation of an n-type is always equivalent to the n-type itself. Since, in proving coherence, we will always consider monoidal groupoids over a 0-type, we will forget from now on about the 1-truncation.[4]

It is then easy to show that list is a functor.

Lemma 3.36. *The function* list *in (3.35) is a functor, in the sense of Definition 3.23.*

Proof. Given types A and $B : \mathcal{U}$ and a function $f : A \to B$, a monoidal functor $\mathsf{list}(f) : \mathsf{MonGpd}(\mathsf{list}(A), \mathsf{list}(B))$ can be produced with underlying function defined by induction:

$$\mathsf{list}(f)(\mathsf{nil}) :\equiv \mathsf{nil},$$
$$\mathsf{list}(f)(x :: l) :\equiv f(x) :: \mathsf{list}(f)(l).$$

We can then define $\mathsf{list}(f)_0 :\equiv \mathsf{refl}_{\mathsf{nil}} : \mathsf{nil} = \mathsf{nil}$; moreover, a function

$$\mathsf{list}(f)_2 : \Pi\,(l_1, l_2 : \mathsf{list}(X))\,.\,\mathsf{list}(f)(l_1) \mathbin{+\!\!+} \mathsf{list}(f)(l_2) = \mathsf{list}(f)(l_1 \mathbin{+\!\!+} l_2)$$

can be defined by induction on l_1:

$$\begin{aligned}
\mathsf{list}(f)_2(\mathsf{nil}, l_2) &: \mathsf{list}(f)(l_2) = \mathsf{list}(f)(l_2) && \text{by refl,}\\
\mathsf{list}(f)_2(x :: l_1, l_2) &: \mathsf{list}(f)(x :: l_1) \mathbin{+\!\!+} \mathsf{list}(f)(l_2)\\
&\equiv f(x) :: \mathsf{list}(f)(l_1) \mathbin{+\!\!+} \mathsf{list}(f)(l_2)\\
&= f(x) :: \mathsf{list}(f)(l_1 \mathbin{+\!\!+} l_2) && \text{by } [f(x) :: -](\mathsf{list}(f)_2(l_1, l_2))\\
&\equiv \mathsf{list}(f)(x :: l_1 \mathbin{+\!\!+} l_2).
\end{aligned}$$

The families of 2-paths $\mathsf{list}(f)_\alpha$ and $\mathsf{list}(f)_\rho$ are given by induction on the leftmost list in a way completely similar to the proof in Fig. 3.13, while $\mathsf{list}(f)_\lambda$ is a family of

[4] In the formalization, we only consider (free) functors $\mathsf{Set} \to \mathsf{MonGpd}$, so this issue is avoided.

trivial 2-paths. The verification that list respects identity and composition of functions is straightforward. For the identity, we need a monoidal natural isomorphism

$$\mathsf{MonFun}_{\mathsf{list}(X),\mathsf{list}(X)}(\mathsf{list}(\mathsf{id}_X), \mathsf{id}_{\mathsf{list}(X)});$$

its underlying homotopy $\theta_{\mathsf{id}} : \mathsf{list}(\mathsf{id}_X) \sim \mathsf{id}_{\mathsf{list}(X)}$ is defined by list induction:

$$\begin{aligned} \theta_{\mathsf{id}}(\mathsf{nil}) &:\equiv \quad \mathsf{refl}_{\mathsf{nil}} \quad : \quad \mathsf{list}(\mathsf{id}_X)(\mathsf{nil}) = \mathsf{id}_{\mathsf{list}(X)}(\mathsf{nil}); \\ \theta_{\mathsf{id}}(x :: l) &:\equiv [x :: -](\theta_{\mathsf{id}}(l)) \; : \; \mathsf{list}(\mathsf{id}_X)(x :: l) = \mathsf{id}_{\mathsf{list}(X)}(x :: l), \end{aligned}$$

for every $x : X$ and $l : \mathsf{list}(X)$. The diagrams in Fig. 3.6 commute: the triangle trivially; the square by induction on the leftmost list. That list respects composition is done similarly. □

If list is free, then Theorem 3.26 is immediately established by Lemma 3.27. In order to show that list is free, we will use another functor as comparison.

3.5 A Free Functor to Monoidal Groupoids

We will use HITs to define a functor $\mathsf{FMG} : \mathcal{U} \to \mathsf{MonGpd}$ so that the resulting construction contains the proof of freeness in its elimination principle.

Definition 3.37 (FMG)**.** Given a type $X : \mathcal{U}$, we define the ap-recursive, 1-truncated HIT $\mathsf{FMG}(X)$ with the following constructors:

$$\begin{aligned} \mathsf{FMG}(X) ::= {} & e : \mathsf{FMG}(X) \mid \iota : X \to \mathsf{FMG}(X) \mid \otimes : \mathsf{FMG}(X) \to \mathsf{FMG}(X) \to \mathsf{FMG}(X) \\ & \mid \alpha : \Pi\,(a, b, c : \mathsf{FMG}(X))\,.\,(a \otimes b) \otimes c = a \otimes (b \otimes c) \\ & \mid \lambda : \Pi\,(b : \mathsf{FMG}(X))\,.\,e \otimes b = b \mid \rho : \Pi\,(a : \mathsf{FMG}(X))\,.\,a \otimes e = a \\ & \mid \pentagon : \ldots \mid \triangledown : \ldots \\ & \mid T : \mathsf{IsHGpd}(\mathsf{FMG}(X)), \end{aligned}$$

where $\pentagon$ and $\triangledown$ are families of 2-path constructors as in (3.12) and (3.13), corresponding to the coherence diagrams of a monoidal groupoid displayed in Fig. 3.1 earlier in this chapter (compare with the definition of $\mathsf{FM}(X)$ in (3.3)). We use subscripts for the arguments of the 1- and 2-constructors of $\mathsf{FMG}(X)$, as in, e.g., $\alpha_{a,b,c}$.

The elimination principle of $\mathsf{FMG}(X)$ follows the scheme given in Section 2.6. The ap-recursivity (see Example 2.144) appears with respect to the constructor $\otimes$ in the coherence diagrams: for example, one of the "sides" of the pentagon $\pentagon_{a,b,c,d}$ is $\alpha_{a,b,c} \otimes \mathsf{refl}_d$, which is shorthand for $[- \otimes -](\alpha_{a,b,c}, \mathsf{refl}_d)$.

It is trivial to see that $\mathsf{FMG}(X)$ is a monoidal groupoid: T guarantees that it is a groupoid, while the monoidal structure is entirely given by its constructors. The resulting construction is also functorial, as proved below.

Lemma 3.38. *The function* $\mathsf{FMG} : \mathcal{U} \to \mathsf{MonGpd}$ *given by*

$$\mathsf{FMG} :\equiv (X \mapsto \langle\langle \mathsf{FMG}(X), T\rangle, e, \otimes, \alpha, \lambda, \rho, \pentagon, \triangledown\rangle)$$

is a functor, in the sense of Definition 3.23.

Proof. Let $f : X \to Y$ be a function of types. A monoidal functor

$$\mathsf{FMG}(f) : \mathsf{MonGpd}(\mathsf{FMG}(X), \mathsf{FMG}(Y))$$

is produced as follows. The underlying function $\mathsf{FMG}(f) : \mathsf{FMG}(X) \to \mathsf{FMG}(Y)$ is defined by the (non-dependent) elimination rule of $\mathsf{FMG}(X)$, by sending the constructors of $\mathsf{FMG}(X)$ to those of $\mathsf{FMG}(Y)$:

$$\mathsf{FMG}(f) :\equiv \mathsf{rec}_{\mathsf{FMG}}(\mathsf{FMG}(Y), e, \iota \circ f, \otimes, \alpha, \lambda, \rho, \pentagon, \triangledown, T),$$

i.e. sending the unit and the product of $\mathsf{FMG}(X)$ to those of $\mathsf{FMG}(Y)$, terms $\iota(x) : \mathsf{FMG}(X)$ to $\iota(f(x)) : \mathsf{FMG}(Y)$ for every $x : X$, and letting the other constructors of $\mathsf{FMG}(Y)$ provide the required paths, 2-paths and 1-truncation. Then the paths:

$$\begin{aligned} \mathsf{FMG}(f)_0 &: e = \mathsf{FMG}(f)(e), \\ \mathsf{FMG}(f)_2(a,b) &: \mathsf{FMG}(f)(a \otimes b) = \mathsf{FMG}(f)(a) \otimes \mathsf{FMG}(f)(b) \end{aligned}$$

can be defined as identity paths; hence, the 2-paths $\mathsf{FMG}(f)_\alpha$, $\mathsf{FMG}(f)_\lambda$ and $\mathsf{FMG}(f)_\rho$ filling the diagrams in Fig. 3.5 reduce to proving that

$$\begin{aligned} [\mathsf{FMG}(f)](\alpha(a,b,c)) &= \alpha(\mathsf{FMG}(f)(a), \mathsf{FMG}(f)(b), \mathsf{FMG}(f)(c)), \\ [\mathsf{FMG}(f)](\lambda(b)) &= \lambda(\mathsf{FMG}(f)(b)), \\ [\mathsf{FMG}(f)](\rho(a)) &= \rho(\mathsf{FMG}(f)(a)); \end{aligned}$$

we have these by the computation rules of the elimination rule of $\mathsf{FMG}(X)$.

We are left to show that FMG preserves identity and composition of functions, i.e., to find monoidal natural isomorphisms

$$\begin{aligned} \mathsf{FMG}_{\mathsf{id}} &: \mathsf{MonFun}_{\mathsf{FMG}(X),\mathsf{FMG}(X)} \left(\mathsf{FMG}(\mathsf{id}_X), \mathsf{id}_{\mathsf{FMG}(X)}\right), \\ \mathsf{FMG}_\circ(f,g) &: \mathsf{MonFun}_{\mathsf{FMG}(X),\mathsf{FMG}(Z)} \left(\mathsf{FMG}(g) \circ \mathsf{FMG}(f), \mathsf{FMG}(g \circ f)\right), \end{aligned}$$

for $X, Y, Z : \mathcal{U}$, $f : X \to Y$ and $g : Y \to Z$. For $\mathsf{FMG}_{\mathsf{id}}$, the underlying homotopy

$$\theta_{\mathsf{id}} : \mathsf{rec}_{\mathsf{FMG}}(\mathsf{FMG}(X), e, \iota, \otimes, \alpha, \lambda, \rho, \pentagon, \triangledown, T) \sim \mathsf{id}_{\mathsf{FMG}(X)}$$

is defined by induction, using the scheme for elimination of 1-truncated HITs into families of paths in a groupoid. We need to provide:

- $\theta_{\mathsf{id}}(e) : e = e$, given by refl_e;
- $\theta_{\mathsf{id}}(\iota(x)) : \iota(x) = \iota(x)$ for every $x : X$, given by $\mathsf{refl}_{\iota(x)}$;
- $\theta_{\mathsf{id}}(a \otimes b) : \mathsf{rec}_{\mathsf{FMG}}(\ldots)(a) \otimes \mathsf{rec}_{\mathsf{FMG}}(\ldots)(b) = a \otimes b$ for every $a, b : \mathsf{FMG}(X)$, given inductive hypotheses $\theta(a)$ and $\theta(b)$; this can then be defined as $\theta(a) \otimes \theta(b)$;
- 2-paths for α, λ and ρ, which are obtained by naturality of associativity and unitality, together with the computation rules of the elimination rule of $\mathsf{FMG}(X)$, as shown by way of example for α in Fig. 3.14.

With these definitions, the diagrams in Fig. 3.6 defining a monoidal natural isomorphism do not require induction on terms in $\mathsf{FMG}(X)$ to be shown to commute, as they trivially do so. The monoidal natural isomorphism $\mathsf{FMG}_\circ(f, g)$ is provided in a completely similar fashion. □

We will proceed to prove that FMG is a free functor by fulfilling, in the following definitions and lemmata, all the conditions listed in Definition 3.24.

Definition 3.39. We define a function

$$\phi : \Pi\,(X : \mathcal{U})\,.\,\Pi\,(M : \mathsf{MonGpd})\,.\,\mathsf{MonGpd}(\mathsf{FMG}(X), M) \to X \to M.$$

Given $X : \mathcal{U}$, $M : \mathsf{MonGpd}$ and a monoidal functor $G : \mathsf{MonGpd}(\mathsf{FMG}(X), M)$, we can produce a function $\phi_{X,M}(G) : X \to M$ by precomposing the underlying function $G : \mathsf{FMG}(X) \to M$ of the monoidal functor G with the constructor $\iota : X \to \mathsf{FMG}(X)$; that is,

$$\phi_{X,M}(G) :\equiv G \circ \iota. \tag{3.40}$$

Lemma 3.41. *The function $\phi_{X,M}$ in Definition 3.39 is natural in M.*

Proof. Given a monoidal functor $H : \mathsf{MonGpd}(M, N)$, we have

$$H \circ \phi_{X,M}(G) \equiv H \circ (G \circ \iota) \equiv (H \circ G) \circ \iota \equiv \phi_{X,N}(H \circ G),$$

so the diagram in Fig. 3.7a commutes judgmentally (hence pointwise) and $\phi_{X,M}$ is natural in M. □

Definition 3.42. We define a function

$$\psi : \Pi\,(X : \mathcal{U})\,.\,\Pi\,(M : \mathsf{MonGpd})\,.\,(X \to M) \to \mathsf{MonGpd}(\mathsf{FMG}(X), M).$$

Given $X : \mathcal{U}$, $M : \mathsf{MonGpd}$ and a function $g : X \to M$, we can produce a monoidal functor $\psi_{X,M}(g) : \mathsf{MonGpd}(\mathsf{FMG}(X), M)$ in the following way:

- the underlying function $\psi_{X,M}(g) : \mathsf{FMG}(X) \to M$ is defined by the elimination rule of $\mathsf{FMG}(X)$ as

$$\psi_{X,M}(g) :\equiv \mathsf{rec}_{\mathsf{FMG}}(M, e_M, g, \otimes_M, \alpha_M, \lambda_M, \rho_M, \pentagon_M, \triangledown_M, T_M) \tag{3.43}$$

  sending the constructors of $\mathsf{FMG}(X)$ to the components of the monoidal structure of M, and terms $\iota(x) : \mathsf{FMG}(X)$ to $g(x) : M$ for every $x : X$;

- the paths

$$\psi_{X,M}(g)_0 : e_M = \psi_{X,M}(g)(e),$$
$$\psi_{X,M}(g)_2(a,b) : \psi_{X,M}(g)(a \otimes b) = \psi_{X,M}(g)(a) \otimes_M \psi_{X,M}(g)(b)$$

  are given by refl for every $a, b : \mathsf{FMG}(X)$, since the two sides of each identity compute to the same terms;

- the 2-paths $\psi_{X,M}(g)_\alpha$, $\psi_{X,M}(g)_\lambda$ and $\psi_{X,M}(g)_\rho$ filling the diagrams in Fig. 3.5 are given by the computation rule of $\mathsf{rec}_{\mathsf{FMG}}$; indeed, since $\psi_{X,M}(g)_0$ and $\psi_{X,M}(g)_2$ are identity paths, they reduce to proving that

$$[\psi_{X,M}(g)](\alpha(a,b,c)) = \alpha_M(\psi_{X,M}(g)(a), \psi_{X,M}(g)(b), \psi_{X,M}(g)(c)),$$
$$[\psi_{X,M}(g)](\lambda(b)) = \lambda_M(\psi_{X,M}(g)(b)),$$
$$[\psi_{X,M}(g)](\rho(a)) = \rho_M(\psi_{X,M}(g)(a)),$$

  for every $a, b, c : \mathsf{FMG}(X)$.

Lemma 3.44. *The function $\psi_{X,M}$ in Definition 3.42 is natural in X.*

Proof. If $h : Y \to X$ is a function of types, a monoidal natural isomorphism

$$\theta_\psi : \mathsf{MonFun}_{\mathsf{FMG}(Y),M}(\psi_{X,M}(g) \circ \mathsf{FMG}(h), \psi_{Y,M}(g \circ h))$$

can be given as follows, for every $g : X \to M$. The underlying homotopy between the underlying functions

$$\theta_\psi : \psi_{X,M}(g) \circ \mathsf{FMG}(h) \sim \psi_{Y,M}(g \circ h)$$

is defined by the elimination rule of FMG on families of paths in a groupoid. On 0-constructors, we have:

$$(e') \qquad \psi_{X,M}(g)(\mathsf{FMG}(h)(e)) \equiv \psi_{X,M}(e) \equiv e_M \equiv \psi_{Y,M}(g \circ h)(e),$$
$$(\iota') \qquad \psi_{X,M}(g)(\mathsf{FMG}(h)(\iota(y))) \equiv \psi_{X,M}(g)(\iota(h(y)))$$
$$\equiv g(h(y)) \equiv \psi_{Y,M}(g \circ h)(\iota(y)),$$

$$
\begin{aligned}
(\otimes') \quad \psi_{X,M}(g)(\mathsf{FMG}(h)(a \otimes b)) &\equiv \psi_{X,M}(g)(\mathsf{FMG}(h)(a) \otimes \mathsf{FMG}(h)(b)) \\
&\equiv \psi_{X,M}(g)(\mathsf{FMG}(h)(a)) \otimes_M \psi_{X,M}(g)(\mathsf{FMG}(h)(b)) \\
&= \psi_{Y,M}(g \circ h)(a) \otimes_M \psi_{Y,M}(g \circ h)(b) \\
&\equiv \psi_{Y,M}(g \circ h)(a \otimes b),
\end{aligned}
$$

where the identity in $(\otimes')$ is given by $\theta_\psi(a) \otimes_M \theta_\psi(b)$. The 2-paths corresponding to the requirement (α'), (λ') and (ρ') are given by naturality of α_M, λ_M and ρ_M, together with the computation rules of the elimination principle of FMG; for example, (α') corresponds to a 2-path filling the diagram in Fig. 3.15. The requirements $(\pentagon')$ and $(\triangledown')$ correspond to 3-paths, which are always present in a groupoid (and unique up to homotopy).

The 2-paths $(\theta_\psi)_0$ and $(\theta_\psi)_2$ are trivial. In particular, all the paths in the triangle $(\theta_\psi)_0$ are defined to be identity paths, while for $(\theta_\psi)_2$ we have that $\theta_\psi(a) \otimes_M \theta_\psi(b) \equiv \theta_\psi(a \otimes b)$ for every $a, b : \mathsf{FMG}(Y)$ and the other sides of the square are defined as identity paths. Hence, a monoidal natural isomorphism making the diagram in Fig. 3.7b commute is provided and $\psi_{X,M}$ is natural in X. □

Lemma 3.45. *There is a homotopy $\theta : \phi_{X,M} \circ \psi_{X,M} \sim \mathsf{id}_{(X \to M)}$, for every $X : \mathcal{U}$ and $M : \mathsf{MonGpd}$, and for ϕ and ψ as in Definitions 3.39 and 3.42.*

Proof. For every $g : X \to M$, we have $\phi_{X,M}(\psi_{X,M}(g)) \equiv \psi_{X,M}(g) \circ \iota \equiv g$, so the homotopy is trivially given. □

Lemma 3.46. *There is a monoidal natural isomorphism*

$$\chi : \mathsf{MonFun}_{\mathsf{FMG}(X),M}(\psi_{X,M}(\phi_{X,M}(G)), G)$$

for every $X : \mathcal{U}$, $M : \mathsf{MonGpd}$ and $G : \mathsf{MonGpd}(\mathsf{FMG}(X), M)$, with ϕ and ψ as in Definitions 3.39 and 3.42.

Proof. First of all, we need to provide a homotopy

$$\chi : \psi_{X,M}(\phi_{X,M}(G)) \sim G,$$

between the underlying functions (in $\mathsf{FMG}(X) \to M$). This we can get by the elimination principle of $\mathsf{FMG}(X)$, using again the scheme for elimination of 1-truncated HITs into families of paths in a groupoid. On 0-constructors, we have, for every $x : X$ and $a, b : \mathsf{FMG}(X)$:

$$
\begin{aligned}
&(e') \quad \psi_{X,M}(\phi_{X,M}(G))(e) \equiv e_M = G(e) && \text{by } G_0, \\
&(\iota') \quad \psi_{X,M}(\phi_{X,M}(G))(\iota(x)) \equiv \phi_{X,M}(G)(x) \equiv G(\iota(x)) = G(\iota(x)) && \text{by } \mathsf{refl}_{G(\iota(x))},
\end{aligned}
$$

$$
\begin{aligned}
(\otimes') \quad & \psi_{X,M}(\phi_{X,M}(G))(a \otimes b) \\
& \equiv \psi_{X,M}(\phi_{X,M}(G))(a) \otimes_M \psi_{X,M}(\phi_{X,M}(G))(b) \\
& = G(a) \otimes_M G(b) && \text{by } \chi(a) \otimes_M \chi(b) \\
& = G(a \otimes b) && \text{by } G_2(a, b).
\end{aligned}
$$

The 2-paths α', λ' and ρ' are given by the computation rules of $\psi_{X,M}$, naturality of α_M, λ_M and ρ_M and by G_α, G_λ and G_ρ; Fig. 3.16 shows α', while the other 2-paths are obtained similarly. With this definition of the underlying homotopy, there are trivial 2-paths χ_0 and χ_2, corresponding to the diagrams in Figs. 3.17a and 3.17b, making χ into a monoidal natural isomorphism. □

Corollary 3.47. FMG *is a free functor in the sense of Definition 3.24, so the monoidal groupoid* $\mathsf{FMG}(X)$ *is freely generated by* X*, for every* $X : \mathcal{U}$*.*

Proof. Follows from the lemmata in this section. □

3.6 The Proof of Coherence

This section is devoted to the proof of Theorem 3.26. We will show that, for every 0-type X, there is a monoidal equivalence

$$\mathsf{FMG}(X) \simeq \mathsf{list}(X). \tag{3.48}$$

A consequence of this equivalence is that (the carrier of) $\mathsf{FMG}(X)$ is a 0-type. Indeed, the property of being a 0-type can be transported along the equivalence, either via univalence (see Section 3.7) or by directly obtaining all 2-paths in $\mathsf{FMG}(X)$ from the corresponding ones in $\mathsf{list}(X)$. Coherence will then follow from Lemma 3.27 and Corollary 3.47.

In order to prove that $\mathsf{FMG}(X)$ is a 0-type, it is actually enough to show the much weaker condition stating that $\mathsf{FMG}(X)$ is a *retract* of $\mathsf{list}(X)$ (Definition 2.88), which we will do in Lemma 3.52. However, we find it interesting to construct explicitly a monoidal equivalence (Corollary 3.56).

Throughout this section, X is a 0-type. All definitions involving X are to be read as given uniformly $(\Pi\,(X : \mathsf{Set})\ldots)$.

Definition 3.49. We define a monoidal functor

$$K : \mathsf{MonGpd}(\mathsf{FMG}(X), \mathsf{list}(X)).$$

The underlying function $K : \mathsf{FMG}(X) \to \mathsf{list}(X)$ is given by the elimination principle of $\mathsf{FMG}(X)$, sending $\iota(x)$ to $x :: \mathsf{nil}$ for every $x : X$, and the monoidal structure of $\mathsf{FMG}(X)$ to the one of $\mathsf{list}(X)$. Precisely, $K(e) :\equiv \mathsf{nil}$ and $K(a \otimes b) :\equiv K(a) +\!\!+ K(b)$ for every $a, b : \mathsf{FMG}(X)$, while the families of paths and 2-paths α', λ', ρ', $\pentagon'$ and $\triangledown'$, required by the elimination principle are given by the monoidal structure of $\mathsf{list}(X)$, discussed in Section 3.4. Finally, the requirement about $\mathsf{list}(X)$ being a 1-type is fulfilled by virtue of Lemma 3.27.

The paths $K_0 : \mathsf{nil} = K(e)$ and $K_2(a, b) : K(a) +\!\!+ K(b) = K(a \otimes b)$ for $a, b : \mathsf{FMG}(X)$ are then given by identity paths; in this way, the required 2-paths K_α, K_λ and K_ρ reduce to exhibiting terms in the following identity types:

$$\alpha_{\mathsf{list}} = [K](\alpha), \qquad \lambda_{\mathsf{list}} = [K](\lambda), \qquad \rho_{\mathsf{list}} = [K](\rho);$$

these are given by the computation rules of K.

Definition 3.50. We define a monoidal functor

$$J : \mathsf{MonGpd}(\mathsf{list}(X), \mathsf{FMG}(X)).$$

The underlying function $J : \mathsf{list}(X) \to \mathsf{FMG}(X)$ is defined by induction on lists:

$$J(\mathsf{nil}) :\equiv e, \qquad J(x :: l) :\equiv \iota(x) \otimes J(l).$$

A path $J_0 : e = J(\mathsf{nil})$ is then given by refl_e, while, given $l_1, l_2 : \mathsf{list}(X)$, a path $J_2(l_1, l_2) : J(l_1) \otimes J(l_2) = J(l_1 +\!\!+ l_2)$ can be produced by induction on l_1:

$$\begin{aligned}
J_2(\mathsf{nil}, l_2) \;:\; & J(\mathsf{nil}) \otimes J(l_2) \equiv e \otimes J(l_2) = J(l_2) \equiv J(\mathsf{nil} +\!\!+ l_2) && \text{by } \lambda;\\
J_2(x :: l, l_2) \;:\; & J(x :: l) \otimes J(l_2) \equiv (\iota(x) \otimes J(l)) \otimes J(l_2) \\
& = \iota(x) \otimes (J(l) \otimes J(l_2)) && \text{by } \alpha \\
& = \iota(x) \otimes J(l +\!\!+ l_2) && \text{by } \mathsf{refl}_{\iota(x)} \otimes J_2(l, l_2) \\
& \equiv J(x :: l +\!\!+ l_2).
\end{aligned}$$

We now need to provide families of 2-paths J_α, J_λ and J_ρ as in Fig. 3.5. With the given definitions of J_0 and J_2, and since $+\!\!+$ satisfies left unitality judgmentally, we can easily find 2-paths $J_\lambda(l)$ for every $l : \mathsf{list}(X)$, as the sought diagram (Fig. 3.18) is trivial. Moreover, for every $p : l_1 =_{\mathsf{list}(X)} l_2$ and $x : X$, we have a 2-path

$$[J]([x :: -](p)) = \mathsf{refl}_{\iota(x)} \otimes [J](p), \tag{3.51}$$

by induction on p. This, together with the coherence diagrams and naturality of associativity and unitality, allows us to define the families of 2-paths $J_\rho(l)$ and $J_\alpha(l_1, l_2, l_3)$ by list elimination (on the first argument for J_α), as shown in Fig. 3.19 and Fig. 3.20 respectively.

In the following lemma, we will show that there is a monoidal natural isomorphism η between the identity monoidal functor $\mathsf{id}_{\mathsf{FMG}(X)}$ and the composition $J \circ K$; the underlying homotopy in η alone proves that $\mathsf{FMG}(X)$ is a retract of $\mathsf{list}(X)$.

Lemma 3.52. *There is a monoidal natural isomorphism*

$$\eta : \mathsf{MonFun}_{\mathsf{FMG}(X),\mathsf{FMG}(X)}(\mathsf{id}, J \circ K).$$

Proof. The underlying homotopy $\eta : \mathsf{id} \sim J \circ K$ is given by the elimination principle of $\mathsf{FMG}(X)$, adopting the scheme for elimination in a type of paths in a groupoid. On 0-constructors we have, for $x : X$ and $a, b : \mathsf{FMG}(X)$:

$$\begin{aligned}
(e') :&\quad e \equiv J(\mathsf{nil}) \equiv J(K(e)) = J(K(e)) && \text{by } \mathsf{refl}_{J(K(e))},\\
(\iota') :&\quad \iota(x) = \iota(x) \otimes e \equiv \iota(x) \otimes J(\mathsf{nil}) && \text{by } \rho^{-1}\\
&\quad \equiv J(x :: \mathsf{nil}) \equiv J(K(\iota(x))),\\
(\otimes') :&\quad a \otimes b = J(K(a)) \otimes J(K(b)) && \text{by } \eta(a) \otimes \eta(b)\\
&\quad = J(K(a) +\!\!+ K(b)) \equiv J(K(a \otimes b)) && \text{by } J_2(K(a), K(b)).
\end{aligned}$$

The requirements α', λ' and ρ' correspond to the diagrams illustrated in Fig. 3.21. We emphasize here the role of the coherence diagrams: α' and ρ' are defined using J_α and J_ρ respectively, both of which employ the coherence diagrams $\pentagon$ and $\triangledown$ (also via the additional coherence diagrams from Lemma 3.16). To complete the proof, a 2-path

$$\eta_0 : \mathsf{id}_0 \cdot \eta(e) = (J \circ K)_0$$

is trivially given, since all terms involved are identity paths. A family of 2-paths

$$\eta_2(a, b) : \mathsf{id}_2(a \otimes b) \cdot \eta(a \otimes b) = \eta(a) \otimes \eta(b) \cdot (J \circ K)_2(a, b)$$

for $a, b : \mathsf{FMG}(X)$ is also trivially given, from the definition of η, since $\mathsf{id}_2(a, b)$ is the identity path and $(J \circ K)_2(a, b) \equiv J_2(K(a), K(b)) \cdot [J](K_2(a, b))$, where $K_2(a, b)$ is the identity path. □

At this point, the proof of coherence then becomes immediate.

Proof of Theorem 3.26. Given a 0-type X, Corollary 3.47 shows that $\mathsf{FMG}(X)$ is the free monoidal groupoid generated by X. For paths $p, q : a =_{\mathsf{FMG}(X)} b$, we have

$$[K](p) =_{(K(a)=K(b))} [K](q)$$

because $\mathsf{list}(X)$ is a set (Lemma 3.27); by path algebra we then get

$$\eta(a) \cdot [J \circ K](p) \cdot \eta(b)^{-1} = \eta(a) \cdot [J \circ K](q) \cdot \eta(b)^{-1};$$

since η is a homotopy, the left-hand side is equal to $[\mathsf{id}](p)$ by Remark 2.105, and hence to p, and similarly the right-hand side is equal to q. Then we have that $p = q$, so $\mathsf{FMG}(X)$ is a set. □

Remark 3.53. In the process of proving coherence, we have produced a technique of normalisation of monoidal expressions in $\mathsf{FMG}(X)$; a normal form in $\mathsf{list}(X)$ is mapped to a monoidal expression in $\mathsf{FMG}(X)$ which contains one and only one instance of the unit e and whose product is right-leaning. An example is shown in Fig. 3.8.

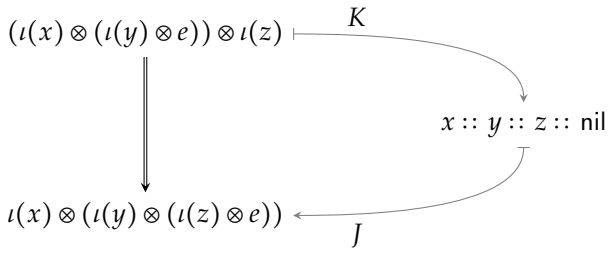


Figure 3.8: Example of normalisation of a monoidal expression in $\mathsf{FMG}(X)$, for $x, y, z : X$.

Although, in order to prove that $\mathsf{FMG}(X)$ is a set, we only needed the underlying homotopy in η, we will show for completeness that K and J are half-adjoint in an equivalence. The equivalence $\mathsf{FMG}(X) \simeq \mathsf{list}(X)$ for every 0-type X can be used to show that $\mathsf{list}(X)$ is free; this will require function extensionality (to construct the homotopy θ of functions between function types in Definition 3.24). Transporting freeness along such a family of equivalences is a tedious exercise and has not been formalized.

Lemma 3.54. *There is a monoidal natural isomorphism*

$$\epsilon : \mathsf{MonFun}_{\mathsf{list}(X),\mathsf{list}(X)}(K \circ J, \mathsf{id}).$$

Proof. The underlying homotopy $\epsilon : K \circ J \sim \mathsf{id}$ is produced by list induction:

$$\begin{aligned} \epsilon(\mathsf{nil}) : &\quad K(J(\mathsf{nil})) \equiv \mathsf{nil} &&= \mathsf{nil} && \text{by } \mathsf{refl}_{\mathsf{nil}}, \\ \epsilon(x :: l) : &\quad K(J(x :: l)) \equiv x :: K(J(l)) &&= x :: l && \text{by } [x :: -](\epsilon(l)). \end{aligned}$$

Since both K_0 and J_0 are identity paths, so is $(K \circ J)_0$. Hence, a 2-path

$$\epsilon_0 : (K \circ J)_0 \cdot \epsilon(\mathsf{nil}) = \mathsf{id}_0$$

is trivially obtained. A family of 2-paths $e_2(l_1, l_2)$ filling the diagram in Fig. 3.22a is obtained by induction on l_1, as shown in Figs. 3.22b and 3.22c, using that

$$[K](\mathsf{refl}_{\iota(x)} \otimes q) = [x :: -]([K](q)) \tag{3.55}$$

for every $q : a =_{\mathsf{FMG}(X)} b$, as proved by induction on q. □

Corollary 3.56. *There is a monoidal equivalence* $\mathsf{FMG}(X) \simeq \mathsf{list}(X)$.

Proof. Follows from the lemmata in this section. □

3.7 Discussion

Comparison to Known Formalizations

The proof of coherence in [BD96], of which a thorough description and formalization can be found in [Bey97], makes implicit use of uniqueness of identity proofs (UIP), specifically when showing coherence for the (discrete) category of normal forms $\mathcal{N}$. The set $\mathrm{N}_{\mathrm{obj}}$ of the objects of $\mathcal{N}$ is defined inductively as the type of lists over a set X; the families of sets $\mathrm{N}_{\mathrm{hom}}(n_1, n_2)$ of arrows between objects n_1 and n_2 are defined inductively, with only the identity arrow as generator:

$$\begin{aligned} \mathrm{N}_{\mathrm{hom}} &: \mathrm{N}_{\mathrm{obj}} \to \mathrm{N}_{\mathrm{obj}} \to \mathsf{Set} \\ &::= \mathrm{N}_{\mathrm{i}} : \Pi\left(n : \mathrm{N}_{\mathrm{obj}}\right).\, \mathrm{N}_{\mathrm{hom}}(n, n). \end{aligned}$$

Finally, the families of sets $\mathrm{N}_{\mathrm{E}}(n_1, n_2, h, h')$ of equalities between arrows h and h' sharing source n_1 and target n_2 are also defined by induction, with reflexivity as the only generator:

$$\begin{aligned} \mathrm{N}_{\mathrm{E}} &: \Pi\left(n_1, n_2 : \mathrm{N}_{\mathrm{obj}}\right).\, \mathrm{N}_{\mathrm{hom}}(n_1, n_2) \to \mathrm{N}_{\mathrm{hom}}(n_1, n_2) \to \mathsf{Set} \\ &::= \mathrm{N}_{\mathrm{ref}} : \Pi\left(n : \mathrm{N}_{\mathrm{obj}}\right).\, \Pi\left(h : \mathrm{N}_{\mathrm{hom}}(n, n)\right).\, \mathrm{N}_{\mathrm{E}}(n, n, h, h). \end{aligned} \tag{3.57}$$

The proof of the statement of coherence, given as

$$\mathrm{N}_{\mathrm{coherence}} : \Pi\left(n_1, n_2 : \mathrm{N}_{\mathrm{obj}}\right).\, \Pi\left(h, h' : \mathrm{N}_{\mathrm{hom}}(n_1, n_2)\right).\, \mathrm{N}_{\mathrm{E}}(n_1, n_2, h, h'),$$

is perfomed by "strong" induction on both h and h', by providing a term

$$\mathrm{N}_{\mathrm{coherence}}(n, n, \mathrm{N}_{\mathrm{i}}(n), \mathrm{N}_{\mathrm{i}}(n)) :\equiv \mathrm{N}_{\mathrm{ref}}(n, \mathrm{N}_{\mathrm{i}}(n)).$$

Without UIP, simultaneous induction on h and h' cannot be performed. Induction on h reduces the goal of coherence to finding a term in $\mathrm{N}_{\mathrm{E}}(n, n, \mathrm{N}_{\mathrm{i}}(n), h')$ for every

$n : \mathrm{N}_{\mathrm{obj}}$ and $h' : \mathrm{N}_{\mathrm{hom}}(n, n)$; further induction on h' cannot be performed, since the elimination principle of $\mathrm{N}_{\mathrm{hom}}$ applies to families

$$C : \Pi\left(n_1, n_2 : \mathrm{N}_{\mathrm{obj}}\right).\, \mathrm{N}_{\mathrm{hom}}(n_1, n_2) \to \mathsf{Set},$$

but the definition

$$C :\equiv (n_1, n_2, f \mapsto \mathrm{N}_{\mathrm{E}}(n, n, \mathrm{N}_{\mathrm{i}}(n), f))$$

is ill-typed, as source and target of $\mathrm{N}_{\mathrm{i}}(n) : \mathrm{N}_{\mathrm{hom}}(n, n)$ and a generic $f : \mathrm{N}_{\mathrm{hom}}(n_1, n_2)$ do not match. This can be solved by redefining the type of the family N_{E} in (3.57):

$$\begin{aligned} \mathrm{N}_{\mathrm{E}} : \Pi\left(n_1, n_2 : \mathrm{N}_{\mathrm{obj}}\right).\, \mathrm{N}_{\mathrm{hom}}(n_1, n_2) \to \mathrm{N}_{\mathrm{hom}}(n_3, n_4) \to \mathsf{Set} \\ :\equiv \mathrm{N}_{\mathrm{ref}} : \Pi\left(n : \mathrm{N}_{\mathrm{obj}}\right).\, \Pi\left(h : \mathrm{N}_{\mathrm{hom}}(n, n)\right).\, \mathrm{N}_{\mathrm{E}}(n, n, h, h), \end{aligned}$$

allowing equality between arrows with varying (*a priori*) source and target.

Employing identity types and HITs in HoTT largely simplifies the definition of a free monoidal groupoid. In [Bey97], the free monoidal category over a set X is defined via:

- an inductive set of objects, corresponding the 0-constructors in our $\mathsf{FMG}(X)$;
- inductive families of arrows, with id, $(- \circ -)$, $(- \otimes -)$, α, α^{-1}, λ, λ^{-1}, ρ and ρ^{-1} as constructors, on which induction is performed when proving the result corresponding to our Lemma 3.52; in our implementation, the groupoid structure of identity types takes care of most of the inductive cases, whereas the cases for α, λ and ρ remain present in the application of the elimination principle of $\mathsf{FMG}(X)$;
- inductive families of equalities between arrows, with a sizeable number of constructors, including: reflexivity, symmetry and transitivity of equality; associativity and unitality of composition; substitution for composition and for the monoidal product; naturality of associativity and unitality of the monoidal product; the interchange law between composition and the monoidal product; composition of associativity and unitality arrows with their inverse; and the coherence diagrams. All but the latter is made redundant in our implementation, as path induction proves everything but the defining diagrams $\pentagon$ and $\triangledown$ of the monoidal structure.

Another feature of our approach is the simplicity in the formulation and proof of freeness of FMG (Section 3.5), omitted in [Bey97].

One final important difference between the two formalizations lies in the design of the normalising functor, which in [Bey97] is made to factor through the type of

functions $\mathsf{list}(X) \to \mathsf{list}(X)$. The argument can be reproduced in our setting: a function $N : \mathsf{FMG}(X) \to \mathsf{list}(X) \to \mathsf{list}(X)$ can be defined via the elimination principle of $\mathsf{FMG}(X)$:

- on 0-constructors, N is defined to send e to the identity function, $\iota(x)$ to the function $(l \mapsto x :: l)$ for every $x : X$, and, recursively, $a \otimes b$ to the composition $N(a) \circ N(b)$, for every $a, b : \mathsf{FMG}(X)$;
- associativity of composition of functions and unitality with respect to the identity function then hold judgmentally, and similarly the coherence diagrams commute;
- $\mathsf{list}(X) \to \mathsf{list}(X)$ is a 0-type, and hence a 1-type (by Remark 2.75).

Subsequently, a function $\mathsf{ev} : \mathsf{list}(X) \to (\mathsf{list}(X) \to \mathsf{list}(X)) \to \mathsf{list}(X)$ can be defined by $\mathsf{ev}(l) :\equiv (f \mapsto f(l))$, evaluating a function to a given term. Normalisation is then achieved by the composition $\mathsf{ev}(\mathsf{nil}) \circ N : \mathsf{FMG}(X) \to \mathsf{list}(X)$. It is possible to show (again using the elimination principle of $\mathsf{FMG}(X)$) that

$$\Pi\,(a : \mathsf{FMG}(X))\,.\,\Pi\,(l : \mathsf{list}(X))\,.\,\mathsf{ev}(l, N(a)) = K(a) +\!\!+ l, \tag{3.58}$$

with $K : \mathsf{FMG}(X) \to \mathsf{list}(X)$ as in Definition 3.49, and hence that $N(a, \mathsf{nil}) = K(a)$ for every $a : \mathsf{FMG}(X)$, by (3.58) followed by an instance of ρ. Thus, the two normalising processes coincide on all monoidal expressions and the resulting (equivalent) proofs of coherence present the same complexity. However, the approach of normalisation "by evaluation" will not easily generalise to coherence for symmetric monoidal groupoids, since composition of functions is not symmetric, so we choose to adopt Definition 3.49 for the normalising functor, directly mapping the monoidal structure of $\mathsf{FMG}(X)$ to that of $\mathsf{list}(X)$.

Univalence and Function Extensionality

With monoidal natural isomorphisms defined as homotopies, our proof does never employ function extensionality. Its use could, however, render the proof shorter: in Lemma 3.27 it is shown that $\mathsf{list}(X)$ is a 0-type whenever X is a 0-type via an "encode-decode" argument; assuming function extensionality, one could see that $\mathsf{list}(X)$ is equivalent to the W-type with labels given by the coproduct $\mathbf{1} + X$ and arities $\mathbf{0}$ for $\mathbf{1}$ and $\mathbf{1}$ for X [see Uni13, Section 5.3]. Since W-types preserve truncation levels [Dan12] and both the labels and the types in the family of arities are 0-types, it follows that $\mathsf{list}(X)$ is also a 0-type.

Theorem 3.26 can be alternatively proved assuming the univalence axiom: the equivalence in (3.48), realized by J and K, provides us with a path $\mathsf{list}(X) = \mathsf{FMG}(X)$,

over which we can transport the proof given in Lemma 3.27 in the family of types $(X \mapsto \mathsf{IsHSet}(X))$ indexed by the universe, to obtain a term in $\mathsf{IsHSet}(\mathsf{FMG}(X))$.

Despite the viable shortcuts described above, we conclude that univalence and function extensionality do not play a key role in the proof of normalisation of monoidal expressions, and hence of coherence for monoidal groupoids. In contrast, we will use function extensionality in Chapter 4 and the full power of univalence in Chapter 5 in order to obtain meaningful results for symmetric monoidal groupoids.

Relationship to Vectors

Assuming function extensionality, the type $\mathsf{list}(X)$ is also equivalent to the type

$$\mathsf{Vec}(X) :\equiv \Sigma\,(n : \mathbb{N})\,.\,([n] \to X)$$

of finite-dimensional *vectors* with coordinates in X. Indeed, a function $f : \mathsf{list}(X) \to \mathsf{Vec}(X)$ can be defined inductively on the list argument:

$$f(\mathsf{nil}) :\equiv \langle 0, \mathsf{rec}_{\mathbf{0}}\rangle, \qquad f(x :: l) :\equiv \langle \mathsf{pr}_1(f(l)) + 1,\ \mathsf{pr}_2(f(l)) + \mathsf{const}_x\rangle,$$

for every $x : X$ and $l : \mathsf{list}(X)$, where $\mathsf{rec}_{\mathbf{0}} : \mathbf{0} \to X$ is the non-dependent version of the elimination principle of the empty type (*ex falso*), while $\mathsf{const}_{(-)} : \mathbf{1} \to X$ is defined as $\mathsf{const}_x(u) :\equiv x$ for every $u : \mathbf{1}$. In the opposite direction, a function $g : \mathsf{Vec}(X) \to \mathsf{list}(X)$ can be given by induction of the first component of the vector (which is a natural number):

$$g\langle 0, v\rangle :\equiv \mathsf{nil} \qquad g\langle n+1, v\rangle :\equiv v(\mathsf{inr}(*)) :: g(v \circ \mathsf{inl}).$$

Proving that f and g are half-adjoint in an equivalence is an easy task; however, as expected, function extensionality is required, since proving a homotopy $f \circ g \sim \mathsf{id}_{\mathsf{Vec}(X)}$ entails showing an *identity* between functions.

The type $\mathsf{Vec}(X)$ can be endowed with a monoidal structure: a monoidal product $\boxplus : \mathsf{Vec}(X) \to \mathsf{Vec}(X) \to \mathsf{Vec}(X)$ is defined so that, for $n_1, n_2 : \mathbb{N}$, $v_1 : [n_1] \to X$ and $v_2 : [n_2] \to X$,

$$\langle n_1, v_1\rangle \boxplus \langle n_2, v_2\rangle :\equiv \langle n_1 + n_2, v_1 \oplus v_2\rangle,$$

where $n_1 + n_2$ is the addition in $\mathbb{N}$, while $v_1 \oplus v_2 : [n_1 + n_2] \to X$ is defined by using the fact that the types $[n_1 + n_2]$ and $[n_1] + [n_2]$ are equivalent, and then by using the elimination principle of the coproduct to build a function $[n_1] + [n_2] \to X$ out of v_1 and v_2. In this way, the functions f and g defined above can be proved to

be half-adjoint in a *monoidal* equivalence, and hence there is a chain of monoidal equivalences

$$\mathsf{FMG}(X) \simeq \mathsf{list}(X) \simeq \mathsf{Vec}(X). \tag{3.59}$$

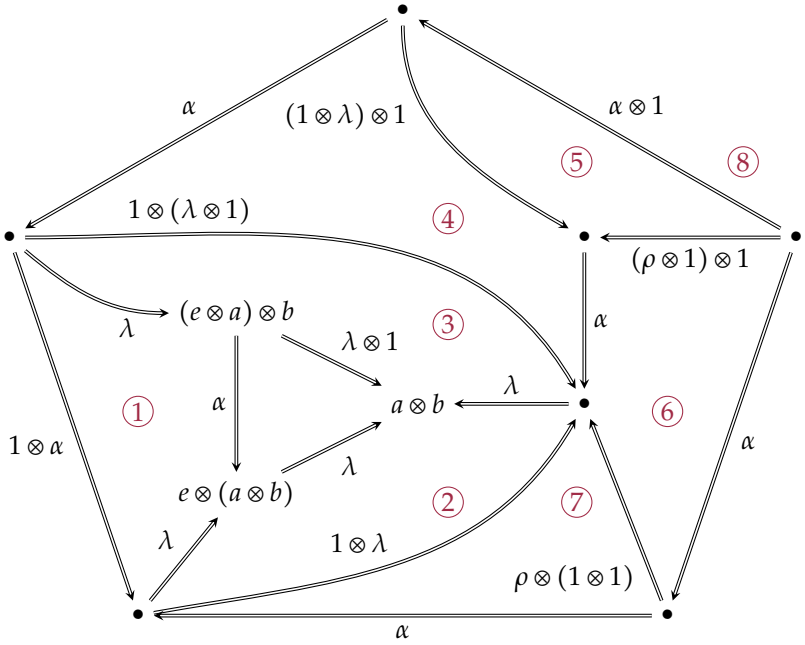
We will not present here the details of this proof, as these will be discussed in a similar setting in Chapter 5.

As the type of finite vectors with coordinates in $\mathbf{1}$ is equivalent to $\mathbb{N}$, we obtain the following chain of monoidal equivalences:

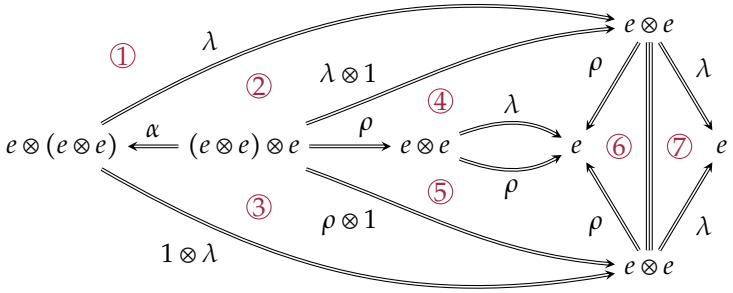
$$\mathsf{FMG}(\mathbf{1}) \simeq \mathsf{list}(\mathbf{1}) \simeq \mathsf{Vec}(\mathbf{1}) \simeq \mathbb{N}, \tag{3.60}$$

although, of course, an equivalence $\mathsf{list}(\mathbf{1}) \simeq \mathbb{N}$ can be obtained more directly. We will investigate a similar chain of equivalences for symmetric monoidal groupoids in Chapter 5.

3.8 Figures in Proofs

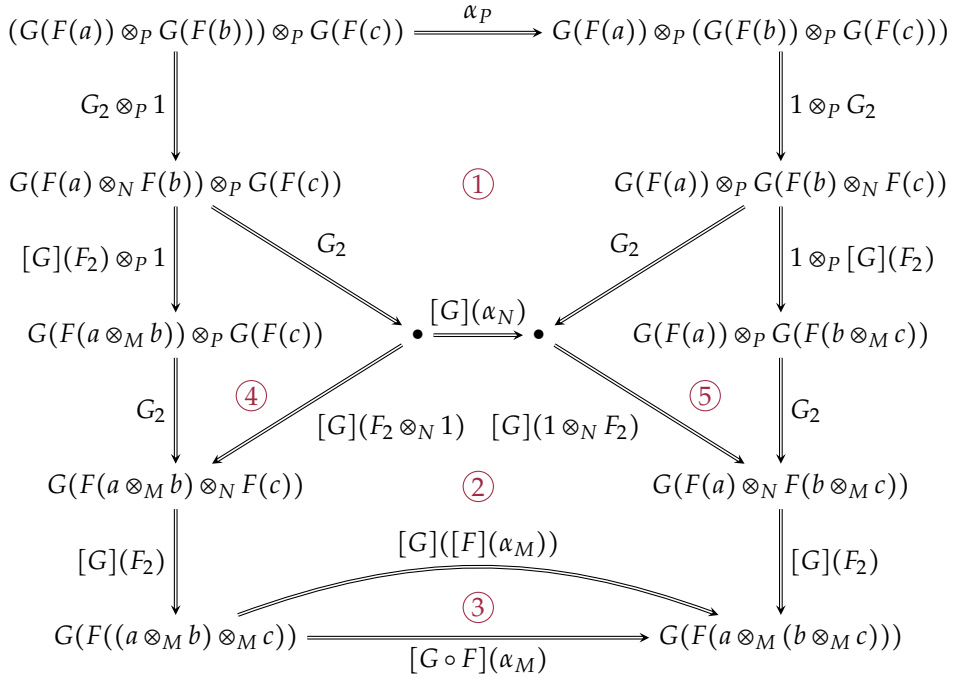


(a) Commutativity of the diagram in Fig. 3.4a, here appearing as the unmarked triangle. The 2-paths (1), (2) and (3) are instances of naturality of λ_M; (4) and (6) are instances of naturality of α_M; (5) and (7) are instances of $\triangledown_M \otimes_M 1$ and $\triangledown_M$ respectively; the outer pentagon (8) is an instance of $\pentagon_M$.

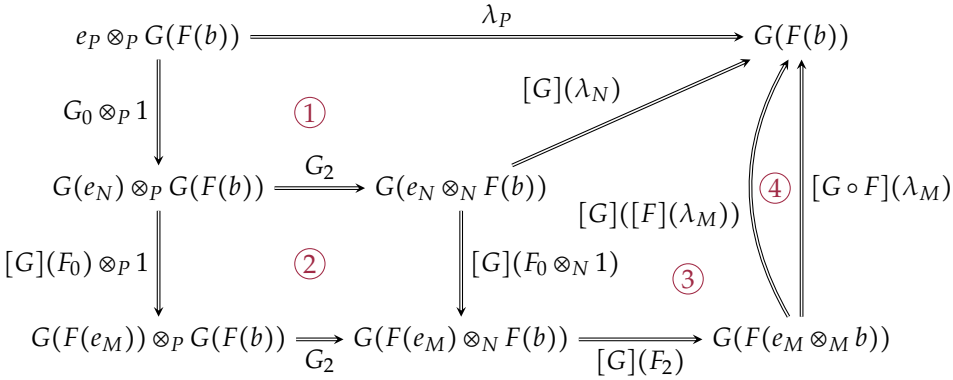


(b) Commutativity of the diagram in Fig. 3.4c, here appearing as the unmarked bigon. The outer square (1) is an instance of naturality of λ_M; the 2-path (2) is the diagram in Fig. 3.4a; (3) is an instance of $\triangledown_M$; (4) and (5) are instances of naturality of ρ_M; (6) and (7) are trivial.

Figure 3.9: Additional coherence diagrams in a monoidal groupoid. We dropped the $-_M$ from the terms in the monoidal structure and replaced some vertices by • for readability.



(a) Derivation of the 2-path $(G \circ F)_\alpha$, after unfolding the definition of $(G \circ F)_2$. The 2-path (1) is G_α; (2) follows from $[[G]](F_\alpha)$; (3), (4) and (5) are instances of functoriality of application of functions.



(b) Derivation of the 2-path $(G \circ F)_\lambda$, after unfolding the definitions of $(G \circ F)_0$ and $(G \circ F)_2$. The 2-path (1) is an instance of G_λ; (2) and (4) are instances of functoriality of application of functions; (3) follows from $[[G]](F_\lambda)$. The derivation of the 2-path $(G \circ F)_\rho$ is done similarly.

Figure 3.10: The composition $G \circ F$ of two monoidal functors G and F is a monoidal functor.

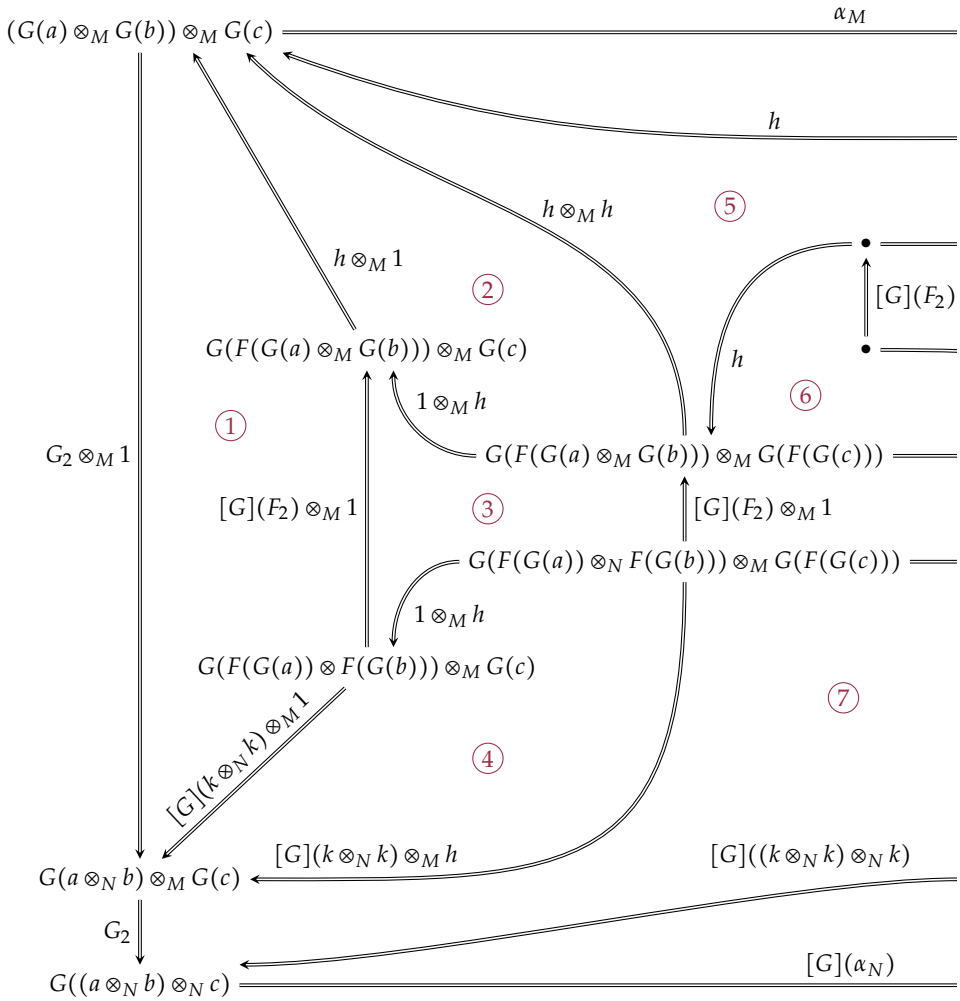
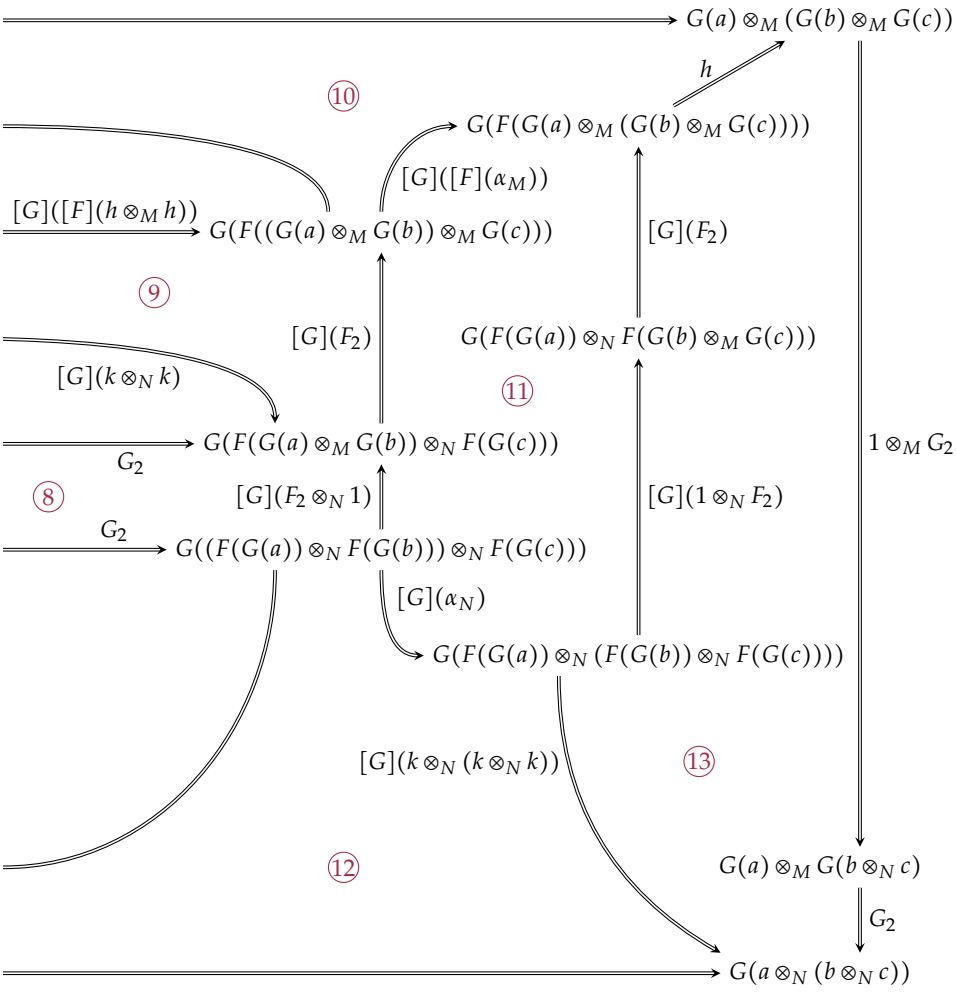


Figure 3.11: Monoidal component G_α of the half-adjoint inverse G of the underlying function of a monoidal functor $F : \mathsf{MonGpd}(M, N)$. The 2-paths (1) and (6) are obtained by the definition of G_2; (2), (3) and (4) are given by path algebra; (5) and (10) are provided by the homotopy h; (7), (8) and (9) are instances of functoriality of application of functions (note that $[G](k) = h$ and $[F](h) = k$); (11) is derived from F_α; (12) is obtained by naturality of α_N; (13) is obtained similarly to the composition of the 2-paths (1)–(9).


$G(a) \otimes_M (G(b) \otimes_M G(c))$
h
10
$G(F(G(a) \otimes_M (G(b) \otimes_M G(c))))$
$[G]([F](\alpha_M))$
$[G]([F](h \otimes_M h))$
$G(F((G(a) \otimes_M G(b)) \otimes_M G(c)))$
$[G](F_2)$
9
$[G](F_2)$
$G(F(G(a)) \otimes_N F(G(b) \otimes_M G(c)))$
$[G](k \otimes_N k)$
11
$G(F(G(a) \otimes_M G(b)) \otimes_N F(G(c)))$
$1 \otimes_M G_2$
G_2
8
$[G](F_2 \otimes_N 1)$
$[G](1 \otimes_N F_2)$
G_2
$G((F(G(a)) \otimes_N F(G(b))) \otimes_N F(G(c)))$
$[G](\alpha_N)$
$G(F(G(a)) \otimes_N (F(G(b)) \otimes_N F(G(c))))$
$[G](k \otimes_N (k \otimes_N k))$
13
12
$G(a) \otimes_M G(b \otimes_N c)$
G_2
$G(a \otimes_N (b \otimes_N c))$

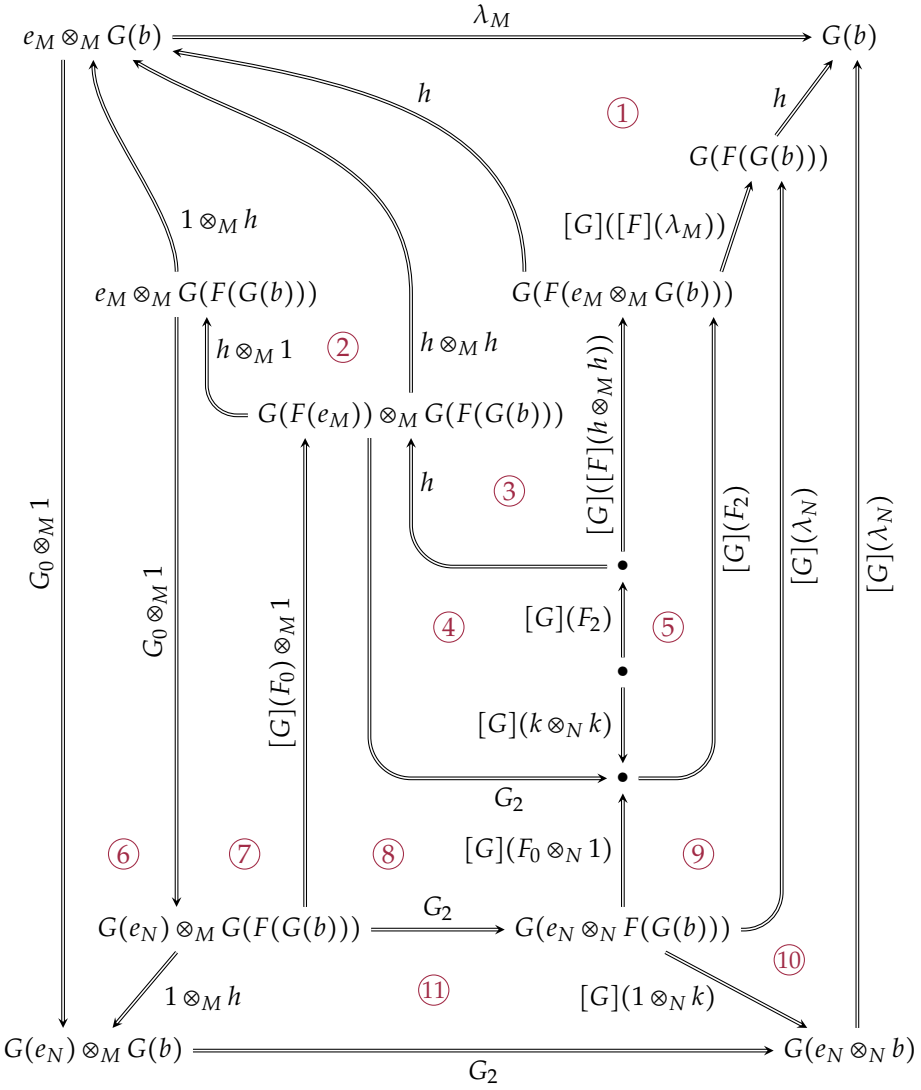
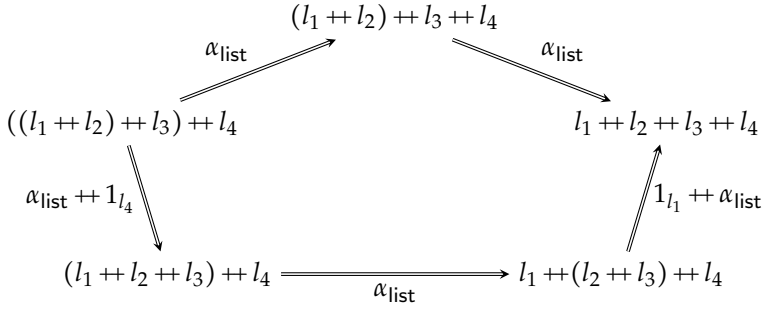
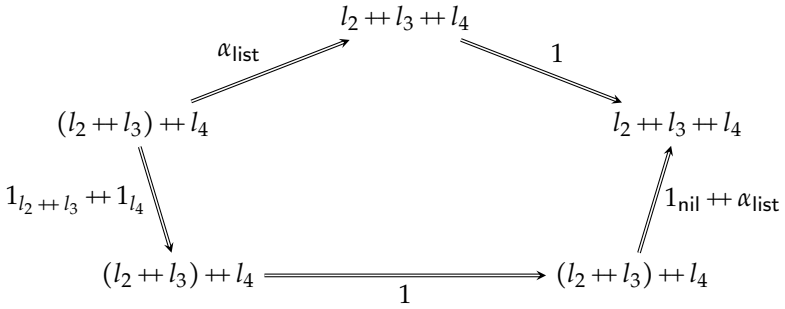


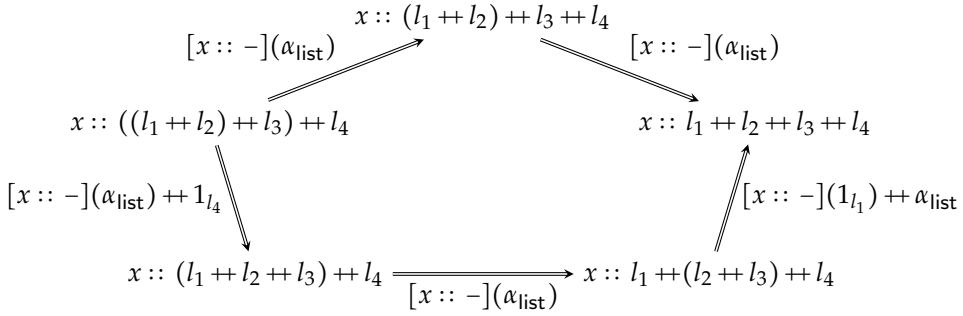
Figure 3.12: Monoidal component G_λ of the half-adjoint inverse G of the underlying function of a monoidal functor $F : \mathsf{MonGpd}(M, N)$. The 2-paths (1) and (3) are provided by the homotopy h; (2) and (6) are given by path algebra; (4) is the definition of G_2; (5), (8) and (11) are instances of functoriality of application of functions (note that $[G](k) = h$ and $[F](h) = k$); (7) is the definition of G_0; (9) is derived from F_λ; (10) is an instance of naturality of λ_N.



(a) The 2-path $\pentagon_{\mathsf{list}}(l_1, l_2, l_3, l_4)$ is produced by induction on l_1.



(b) The 2-path $\pentagon_{\mathsf{list}}(\mathsf{nil}, l_2, l_3, l_4)$. Using (3.33), the diagram becomes trivial.



(c) The 2-path $\pentagon_{\mathsf{list}}(x :: l_1, l_2, l_3, l_4)$, filled using $[[x :: -]](\pentagon_{\mathsf{list}}(l_1, l_2, l_3, l_4))$ recursively, together with (3.34).

Figure 3.13: Coherence pentagon for $+\!\!+$.

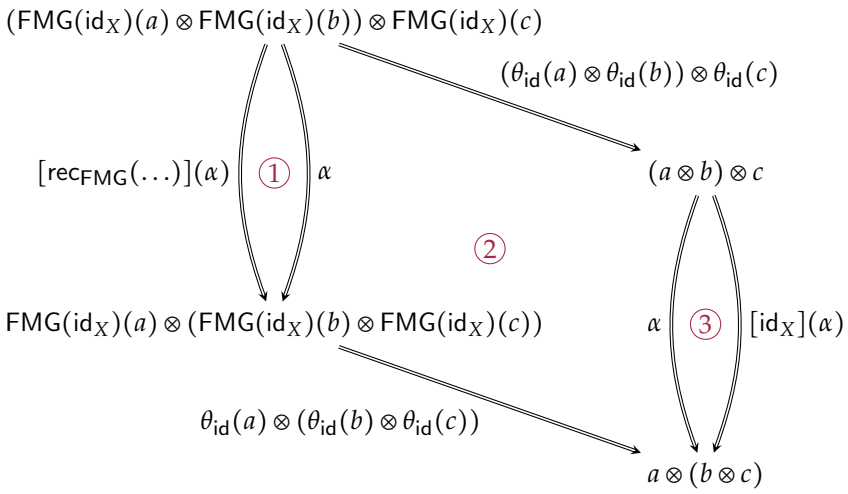


Figure 3.14: FMG, as a functor, respects identity; diagram for α. The 2-path in (1) is given by a computation rule of the elimination principle of $\mathsf{FMG}(X)$; (2) is an instance of naturality of α; (3) is obtained by path algebra. The diagrams for λ and ρ are similarly proved.

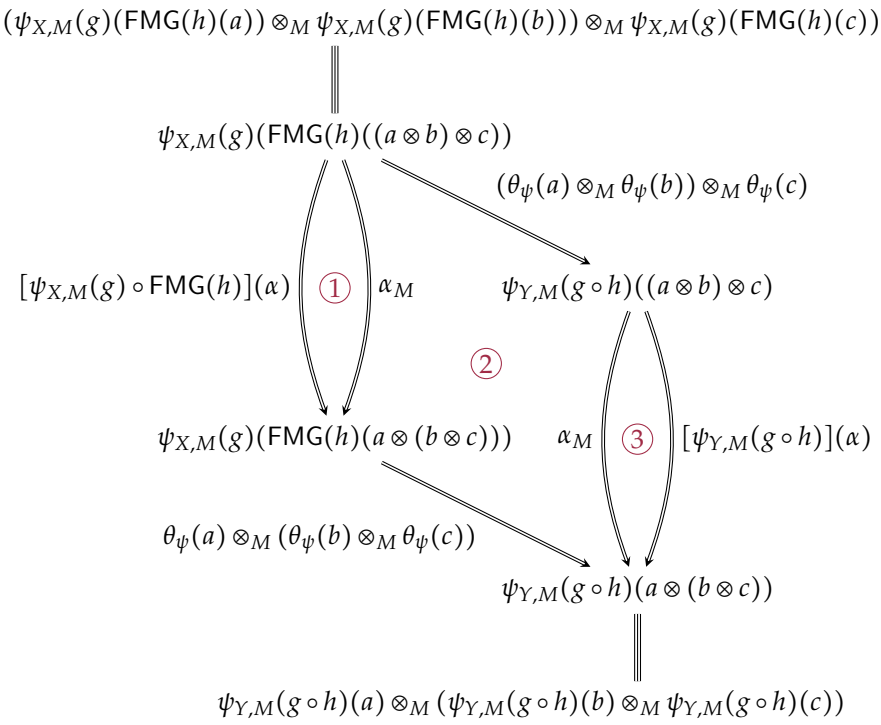


Figure 3.15: The underlying homotopy θ_ψ in the proof of naturality of $\psi_{X,M}$ in X is achieved via the elimination rules of $\psi_{X,M}$ and FMG; these require certain 2-paths in M to be filled, corresponding to the 1-path constructors of $\mathsf{FMG}(Y)$. This figure shows the 2-path for associativity. The 2-paths (1) and (3) are given by the computation rules of $\psi_{X,M}$ and FMG; (2) is filled by naturality of α_M. The terms on the top and bottom of the diagram are judgmentally equal to the top-left and bottom-right corner of the square. The 2-paths in M corresponding to the constructors for unitality are proved similarly.

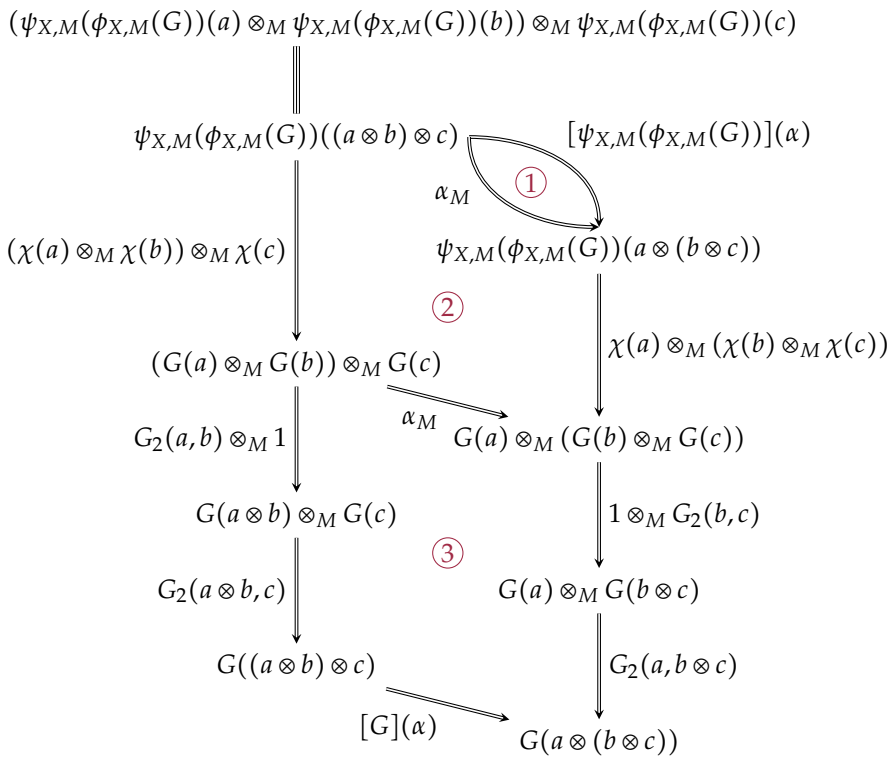


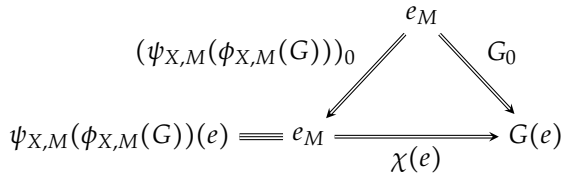
Figure 3.16: The 2-path in M providing α' in the definition of χ using the elimination principle of $\mathsf{FMG}(X)$. The 2-path (1) is given by a computation rule of $\psi_{X,M}$; (2) is an instance of naturality of α_M; (3) is an instance of G_α. The term on the top is judgmentally equal to the top-left corner of (2). The vertical paths correspond to the ones given by $\otimes'$, after (2.64): indeed, there are 2-paths

$$\begin{aligned}\chi((a \otimes b) \otimes c) &\equiv (((\chi(a) \otimes_M \chi(b)) \cdot G_2(a,b)) \otimes_M \chi(c)) \cdot G_2(a \otimes b, c) \\ &= ((\chi(a) \otimes_M \chi(b)) \otimes_M \chi(c)) \cdot (G_2(a,b) \otimes_M 1) \cdot G_2((a \otimes b), c)\end{aligned}$$

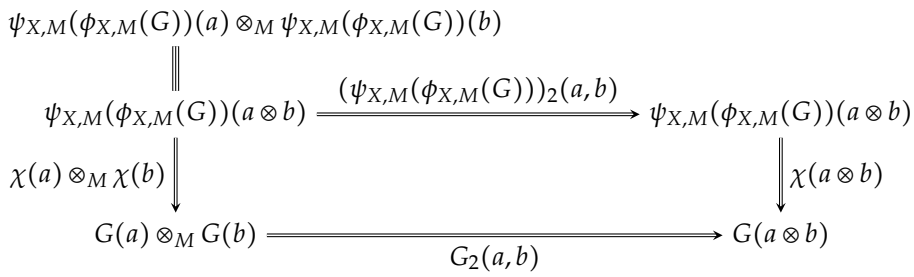
and

$$\begin{aligned}\chi(a \otimes (b \otimes c)) &\equiv \chi(a) \otimes_M ((\chi(b) \otimes_M \chi(c)) \cdot G_2(b,c)) \cdot G_2(a, b \otimes c) \\ &= (\chi(a) \otimes_M (\chi(b) \otimes_M \chi(c))) \cdot (1 \otimes_M G_2(b,c)) \cdot G_2(a, b \otimes c).\end{aligned}$$

The 2-paths for λ' and ρ' are obtained similarly.



(a) The 2-path χ_0 corresponding to the diagram in Fig. 3.17a is trivial, as $(\psi_{X,M}(\phi_{X,M}(G)))_0 \equiv \mathsf{refl}_{e_M}$ by definition and $\chi(e) \equiv G_0$ by computation rule of χ.



(b) For $a, b : \mathsf{FMG}(X)$, the 2-path χ_2 corresponding to the diagram in is also trivial, since by definition

$$(\psi_{X,M}(\phi_{X,M}(G)))_2(a,b) \equiv \mathsf{refl}_{\psi_{X,M}(\phi_{X,M}(G))(a \otimes b)},$$

while $\chi(a \otimes b) \equiv (\chi(a) \otimes_M \chi(b)) \cdot G_2(a \otimes b)$ by computation rule of χ.

Figure 3.17: The 2-paths χ_0 and χ_2 in the definition of χ as a monoidal natural isomorphism.

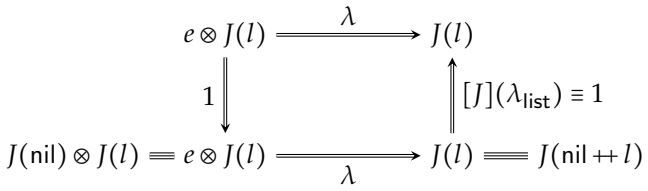
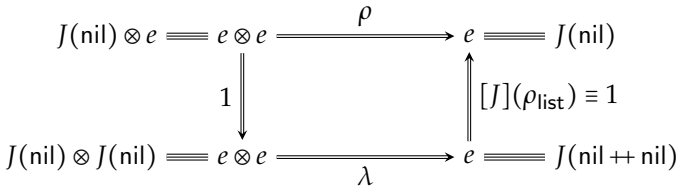
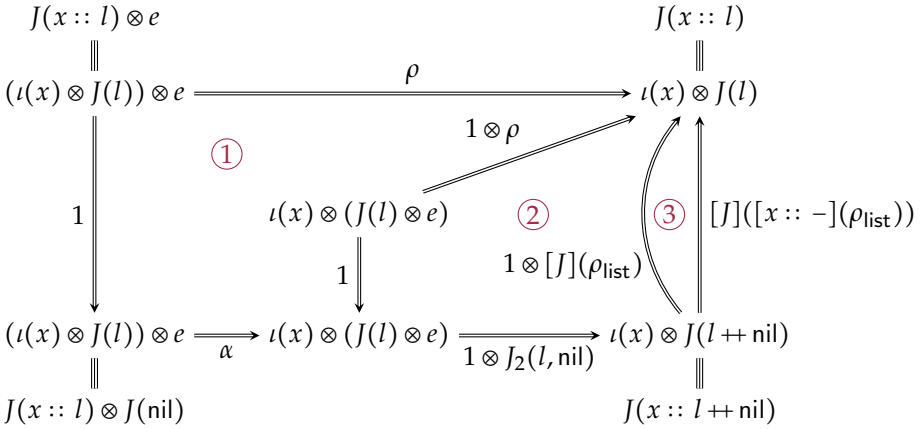


Figure 3.18: Construction of the 2-path $J_\lambda(l)$, after unfolding the definitions of J_0 and J_2. The square is filled by path algebra.

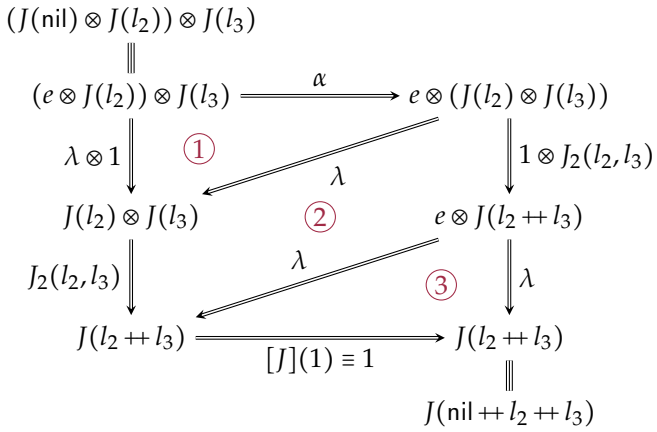


(a) The 2-path $J_\rho(\mathsf{nil})$ in the inductive definition of J_ρ can be obtained by the additional coherence diagram in Fig. 3.4c.

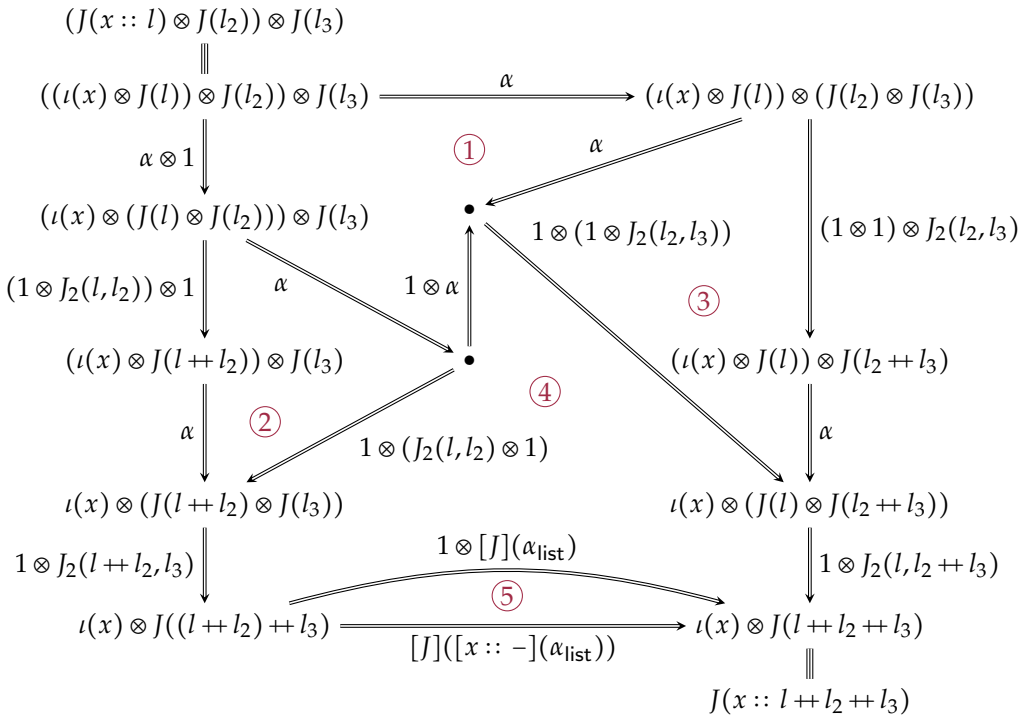


(b) The 2-path $J_\rho(x :: l)$ in the inductive definition of J_ρ. The 2-path (1) is an instance of the additional coherence diagram in Fig. 3.4b; (2) is given recursively by $\mathsf{refl}_{\iota(x)} \otimes J_\rho(l)$; (3) is an instance of (3.51).

Figure 3.19: Construction of the 2-path $J_\rho(l)$ by induction on l, after unfolding the definitions of J_0, J_2 and some path algebra.

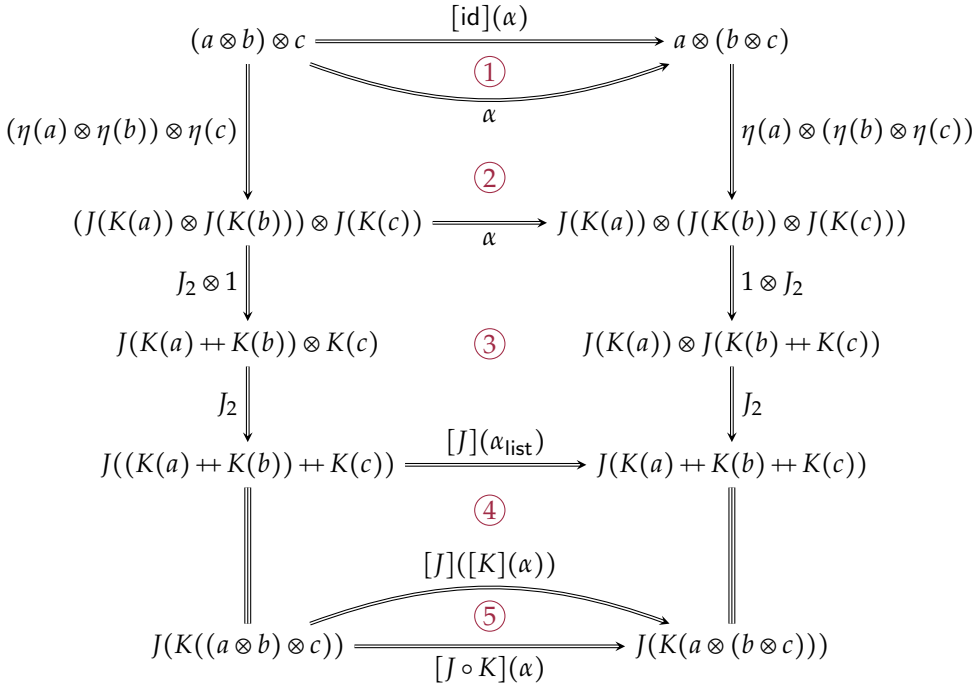


(a) The 2-path $J_\alpha(\mathsf{nil}, l_2, l_3)$ in the inductive definition of J_α. The 2-path (1) is an instance of the additional coherence diagram in Fig. 3.4a; (2) is an instance of naturality of λ; (3) is trivial.



(b) The 2-path $J_\alpha(x :: l, l_2, l_3)$ in the inductive definition of J_α. The 2-path (1) is an instance of $\pentagon$; (2) and (3) are instances of naturality of α; (4) is given recursively by $\mathsf{refl}_{\iota(x)} \otimes J_\alpha(l, l_2, l_3)$; (5) is an instance of (3.51).

Figure 3.20: Construction of the 2-path $J_\alpha(l_1, l_2, l_3)$ by induction on l_1, after unfolding the definition of J_2 and some path algebra.

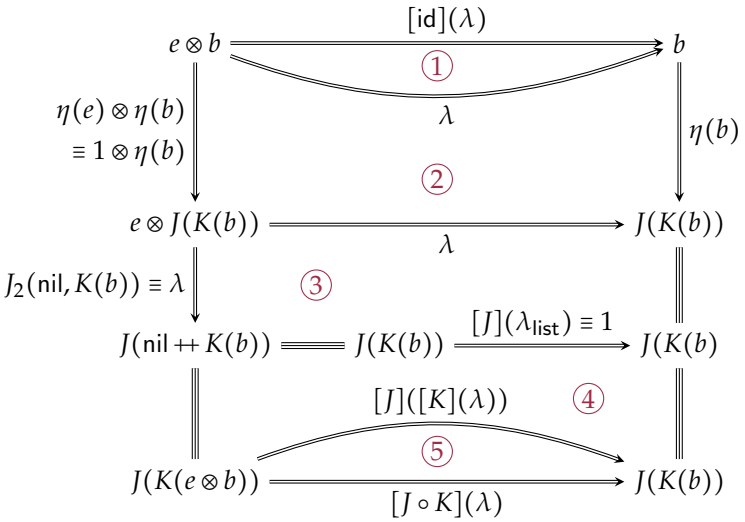


(a) The 2-path for α' in the inductive definition of η. The vertical path on the left is equal to $\eta((a \otimes b) \otimes c)$, since

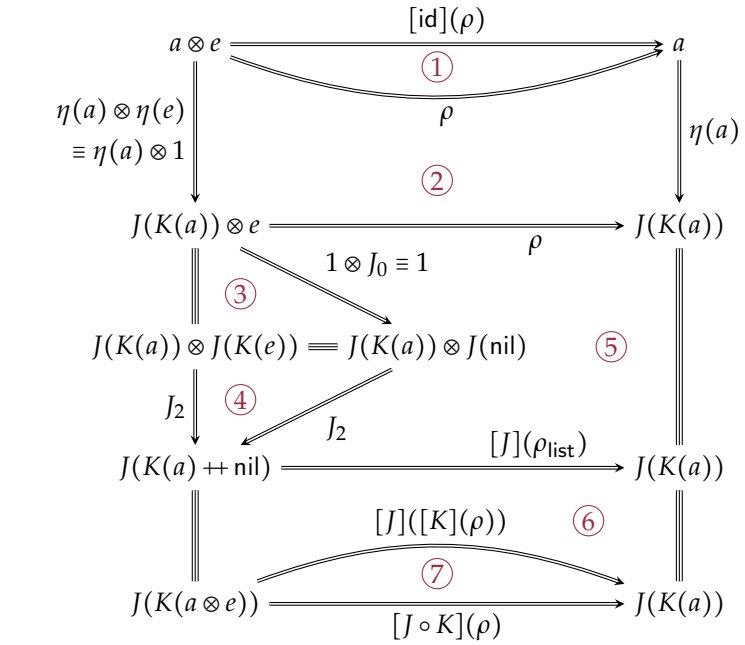
$$\begin{aligned}\eta((a \otimes b) \otimes c) &\equiv (\eta(a \otimes b) \otimes \eta(c)) \cdot J_2(K(a \otimes b), K(c)) \\ &\equiv (((\eta(a) \otimes \eta(b)) \cdot J_2(K(a), K(b))) \otimes \eta(c)) \cdot J_2(K(a) +\!\!+ K(b), K(c)) \\ &= ((\eta(a) \otimes \eta(b)) \otimes \eta(c)) \cdot (J_2(K(a), K(b)) \otimes \mathsf{refl}) \cdot J_2(K(a) +\!\!+ K(b), K(c))\end{aligned}$$

using (2.64); similarly for the vertical path on the right. The 2-paths (1) and (5) are given by path algebra; (2) is an instance of naturality of α; (3) is an instance of J_α; (4) is given by a computation rule of K.

Figure 3.21: The diagrams corresponding to α', λ' and ρ' in the inductive definition of η, after unfolding the definition of $\otimes'$ (vertical sides) and some path algebra.

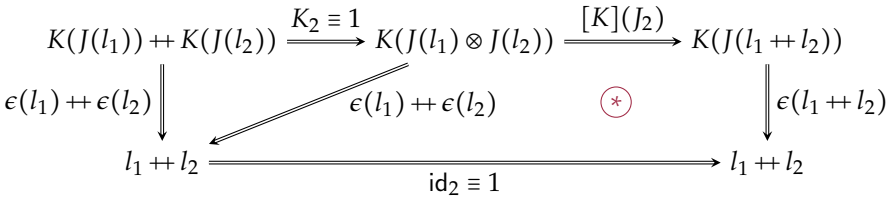


(b) The 2-path for λ' in the inductive definition of η. The 2-paths (1) and (5) are given by path algebra; (2) is an instance of naturality of λ; (3) is trivial, as $J_2(\mathsf{nil}, K(b)) \equiv \lambda$; (4) is given by a computation rule of K.

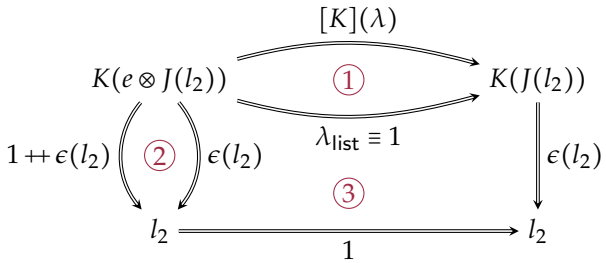


(c) The 2-path for ρ' in the inductive definition of η. The 2-paths (1) and (7) are given by path algebra; (2) is an instance of naturality of ρ; (3) and (4) are trivial; (5) is an instance of J_ρ; (6) is given by a computation rule of K.

Figure 3.21: Continued.

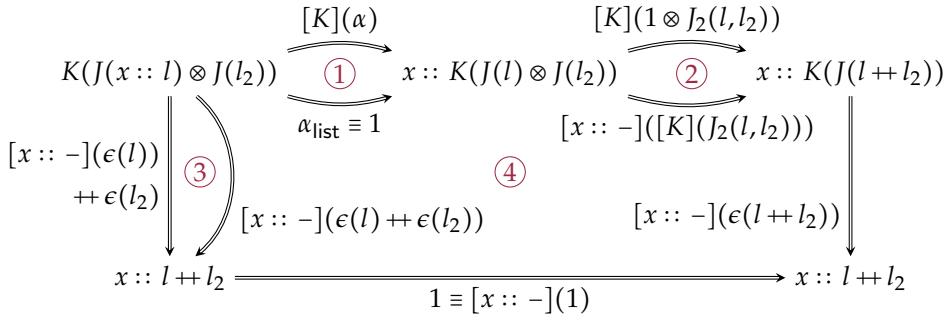


(a) The sought diagram for $\epsilon_2(l_1, l_2)$, filled by induction on l_1, as shown in Figs. 3.22b and 3.22c. Since K_2 is the identity path, we will focus on (*), unfolding in each case the definition of J_2.



(b) The 2-path $\epsilon_2(\mathsf{nil}, l_2)$ in the inductive definition of ϵ_2. The top-left term is judgmentally equal to $\mathsf{nil} +\!\!+ K(J(l_2)) \equiv K(J(l_2))$ and the bottom-left term is judgmentally equal to $\mathsf{nil} +\!\!+ l_2$. The 2-path (1) is given by a computation rule of K; (2) is an instance of (3.33); (3) is trivial.

Figure 3.22: Derivation of the 2-path $\epsilon_2(l_1, l_2)$ for $l_1, l_2 : \mathsf{list}(X)$.



(c) The 2-path $\epsilon_2(x :: l, l_2)$ in the inductive definition of ϵ_2. The 2-path (1) is an instance of a computation rule of K; observe that

$$[K](\alpha) : K(J(x :: l) \otimes J(l_2)) \equiv K((\iota(x) \otimes J(l)) \otimes J(l_2)) = K(\iota(x) \otimes (J(l) \otimes J(l_2))) \\ \equiv x :: K(J(l) \otimes J(l_2))$$

and that

$$\alpha_{\mathsf{list}} : K(J(x :: l) \otimes J(l_2)) \equiv x :: (\mathsf{nil} +\!\!+ K(J(l))) +\!\!+ K(J(l_2)) = x :: \mathsf{nil} +\!\!+ K(J(l)) +\!\!+ K(J(l_2)) \\ \equiv x :: K(J(l) \otimes J(l_2)),$$

is the identity path, since $\alpha_{\mathsf{list}}(x :: \mathsf{nil}, \ldots, \ldots) \equiv [x :: -](\alpha_{\mathsf{list}}(\mathsf{nil}, \ldots, \ldots)) \equiv [x :: -](\mathsf{refl}) \equiv \mathsf{refl}$. The 2-path (2) is an instance of (3.55), where

$$[K](1 \otimes J_2) : x :: K(J(l) \otimes J(l_2)) \equiv K(\iota(x) \otimes (J(l) \otimes J(l_2))) = K(\iota(x) \otimes J(l +\!\!+ l_2)) \equiv x :: K(J(l +\!\!+ l_2)).$$

The 2-path (3) is an instance of (3.34), where the source of the arrows is judgmentally equal to $x :: (K(J(l)) +\!\!+ K(J(l_2)))$; (4) is given recursively by $[[x :: -]](\epsilon_2(l, l_2))$.

Figure 3.22: Continued.

Chapter 4

Coherence for Symmetric Monoidal Groupoids

A natural extension of the result presented in Chapter 3 concerns coherence for *symmetric* monoidal categories. A symmetric monoidal structure on a category extends a monoidal structure by introducing a braiding for the monoidal product, i.e., a natural isomorphism $\tau_{a,b} : a \otimes b \to b \otimes a$, producing involutions that satisfy Yang-Baxter relations (Fig. 4.2). Symmetric monoidal categories also satisfy a theorem of coherence [ML98], stating that they can be strictified up to associativity and unitality, but not symmetry. In particular, *not* every diagram in a free symmetric monoidal category commutes; Fig. 4.1 shows an example of such a diagram. For this reason, a coherence statement for symmetric monoidal categories is harder to formulate than its non-symmetric counterpart, as it involves isolating the class of commutative diagrams from the class of all diagrams in the category.

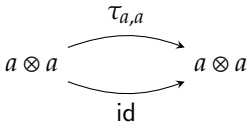


Figure 4.1: A non-commutative diagram in a free symmetric monoidal category.

In this chapter, we will adapt the results presented in Chapter 3 in order to obtain a proof of coherence for symmetric monoidal groupoids via normalisation of symmetric monoidal expressions. As we will see, univalence and function extensionality will be needed in this presentation.

The content of this chapter has been formalized entirely; selected parts of the formalization are featured in Appendix A.2.

4.1 Symmetric Monoidal Groupoids

A symmetric monoidal groupoid is a groupoid endowed with a *symmetric* monoidal structure, which features a monoidal product which is symmetric, and ensuing coherence diagrams. We use the definition and notation for groupoids from the previous chapter (Definition 3.9).

Definition 4.1. A **symmetric monoidal structure** on a type M is the data consisting of a unit term and a product in M and families of paths and 2-paths witnessing 1-coherent associativity, unitality with respect of the unit term, and symmetry of the product; that is, given a type M, we define the type $\mathsf{SymMonStructure}(M)$ as the Σ-type encoding the following data:

- terms $e_M : M$ (unit) and $\otimes_M : M \to M \to M$ (symmetric monoidal product);
- families of paths α_M, λ_M, ρ_M (resp. associativity, left and right unitality) as in Definition 3.11;
- a family of paths $\tau_M : \Pi\,(a, b : M)\,.\,a \otimes_M b = b \otimes_M a$ (symmetry);
- families of 2-paths $\pentagon_M$, $\triangledown_M$ as in Definition 3.11, and $\hexagon_M$, $\diamondsuit_M$ filling the coherence diagrams depicted in Fig. 4.2, i.e.:

$$\begin{aligned}
\hexagon_M : \Pi\,(a, b, c : M)\,.\,&\alpha_M(a, b, c) \cdot \tau_M(a, b \otimes_M c) \cdot \alpha_M(b, c, a) \\
&= (\tau_M(a, b) \otimes_M \mathsf{refl}_c) \cdot \alpha_M(b, a, c) \cdot (\mathsf{refl}_b \otimes_M \tau_M(a, c)), \\
\diamondsuit_M : \Pi\,(a, b : M)\,.\,&\tau_M(a, b) \cdot \tau_M(b, a) = \mathsf{refl}_{a \otimes_M b}.
\end{aligned}$$

As for monoidal groupoids, we define a type of **symmetric monoidal groupoids** as:

$$\mathsf{SymMonGpd} :\equiv \Sigma\,(M : \mathsf{Gpd})\,.\,\mathsf{SymMonStructure}(M)$$

and use the same notation for a symmetric monoidal groupoid $M : \mathsf{SymMonGpd}$ and its carrier.

Remark 4.2. Assuming univalence, the universe $\mathcal{U}$ (though not a 1-type) can be endowed with a symmetric monoidal structure in several ways; for example:

- with the product type former $\times : \mathcal{U} \to \mathcal{U} \to \mathcal{U}$ as symmetric monoidal product and the unit type $\mathbf{1}$ as unit of the structure;
- with the coproduct type former $+ : \mathcal{U} \to \mathcal{U} \to \mathcal{U}$ as symmetric monoidal product and the empty type $\mathbf{0}$ as unit of the structure.

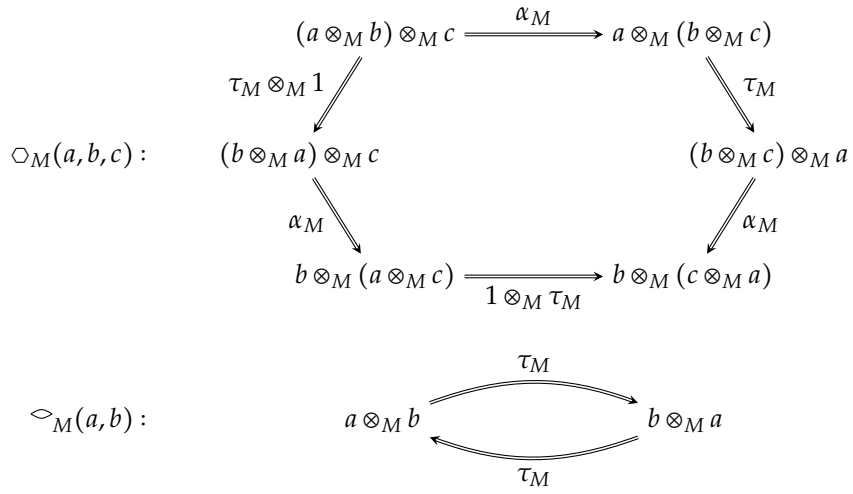


Figure 4.2: Coherence diagrams $\hexagon$ and $\diamond$ in a symmetric monoidal groupoid.

In both cases, the families of paths $\alpha_{\mathcal{U}}$, $\lambda_{\mathcal{U}}$, $\rho_{\mathcal{U}}$ and $\tau_{\mathcal{U}}$ are produced, via univalence, by families of *equivalences* witnessing associativity, unitality (with respect to the unit and the empty type, respectively) and symmetry of the product and coproduct of types, while the coherence diagrams are obtained by "univalence algebra" (Lemma 2.113). The symmetric monoidal structure induced by the coproduct of types is discussed in more detail in Section 5.1; the one induced by the product of types is obtained analogously.

Remark 4.3. The universe $\mathcal{U}^* :\equiv \Sigma\,(X : \mathcal{U})\,.\,X$ of *pointed types* is supposed to find a symmetric monoidal structure seeing the pointed type $\langle \mathbf{2}, \mathsf{yes} \rangle$ as unit and the *smash product* $\wedge : \mathcal{U}^* \to \mathcal{U}^* \to \mathcal{U}^*$ as symmetric monoidal product. This can be defined, for every $\langle A, a_0 \rangle$, $\langle B, b_0 \rangle : \mathcal{U}^*$ as the type given by the HIT

$$\begin{aligned} A \wedge B :\equiv \langle -, - \rangle : A \to B \to (A \wedge B) \;\mid\; &\mathsf{auxl},\ \mathsf{auxr} : A \wedge B \\ \mid\; &\mathsf{gluel} : \Pi\,(a : A)\,.\,\langle a, b_0 \rangle = \mathsf{auxl} \;\mid\; \mathsf{gluer} : \Pi\,(b : B)\,.\,\langle a_0, b \rangle = \mathsf{auxr}, \end{aligned}$$

pointed at $\langle a_0, b_0 \rangle$. The efforts spent in defining a complete symmetric monoidal structure are presented in [vD18, Section 4.3].[1] There, the structural components rely on notions of "pointed equivalences" and "pointed homotopies" rather than paths and 2-paths, similarly to the structures described in Remark 4.2, whose presentation in terms of paths and 2-paths is entirely cosmetic and solely based on established equivalences between types.

[1] Joint work of Floris van Doorn and S. P.

Remark 4.4. There is a projection $\mathsf{SymMonStructure}(M) \to \mathsf{MonStructure}(M)$ for every type M. Moreover, the naturality squares for associativity and unitality (in Lemma 3.15) and all diagrams commuting for a monoidal structure (e.g. Lemma 3.16) do, obviously, commute also for the data given by a symmetric monoidal structure.

As we did for (non-symmetric) monoidality, we will show the proof of commutativity of certain diagrams, which we will encounter in this thesis in the proof of some statements.

Lemma 4.5. *Symmetry in a monoidal structure is natural in both arguments, i.e., given paths $p : a =_M a'$ and $q : b =_M b'$, there is a 2-path filling the diagram in Fig. 4.3.*

Proof. By induction on p and q. □

Lemma 4.6. *The diagrams in Fig. 4.4 commute for every $a, b, c : M$.*

Proof. Commutativity of the diagram in Fig. 4.4a is shown in Fig. 4.12. Commutativity of the diagrams in Fig. 4.4b and Fig. 4.4c is proved by glueing instances of $\hexagon_M$ to, respectively, the diagram in Fig. 4.4a and $\hexagon_M(a, b, c)$ [see also Kel64; JS93]. □

We can extend the definition of monoidal functors to include symmetry.

Definition 4.7. The type of (strong) **symmetric monoidal functors** between two symmetric monoidal groupoids $\langle M, e_M, \otimes_M, \ldots\rangle$ and $\langle N, e_N, \otimes_N, \ldots\rangle$ is defined as the Σ-type encoding the following data:

- a functor $F : M \to N$, a path $F_0 : e_N = F(e_M)$ and a family of paths $F_2 : \Pi\,(a, b : M)\,.\,F(a) \otimes_N F(b) = F(a \otimes_M b)$ as in Definition 3.17;
- families of 2-paths F_α, F_λ, F_ρ as in Definition 3.17 and F_τ corresponding to the diagram in Fig. 4.5 for every $a, b : M$.

We will denote this type by $\mathsf{SymMonGpd}(M, N)$ and equate, in the notation, a symmetric monoidal functor and its underlying functor, as we did for MonGpd.

The identity symmetric monoidal functor $\mathsf{id}_M : \mathsf{SymMonGpd}(M, M)$ and the composition $(G \circ F) : \mathsf{SymMonGpd}(M, P)$ of any two symetric monoidal functors $F : \mathsf{SymMonGpd}(M, N)$ and $G : \mathsf{SymMonGpd}(N, P)$ are defined in the same way as for (non-symmetric) monoidal functors; in particular, the 2-path $(G \circ F)_\tau$ is displayed in Fig. 4.13.

There is no distinction between the notions of monoidal natural isomorphism and its symmetric counterpart, other than the fact that they refer to different classes of functors.

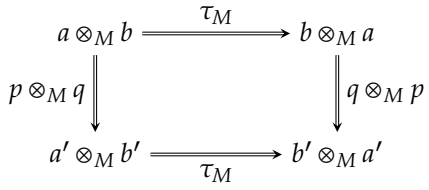


Figure 4.3: Naturality of τ_M.

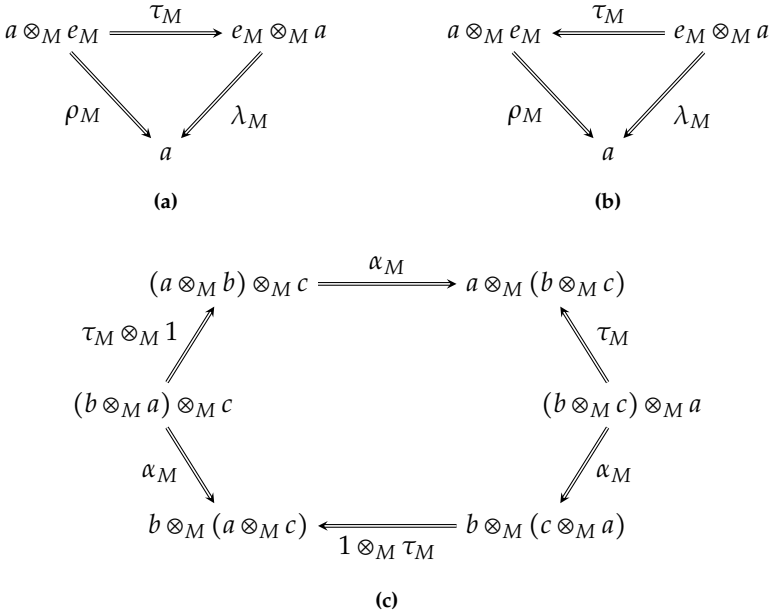


Figure 4.4: Additional coherence diagrams in a symmetric monoidal category.

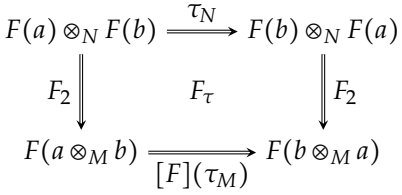


Figure 4.5: Coherence condition for symmetric monoidal functors.

Definition 4.8. The type $\mathsf{SymMonFun}_{M,N}(F, G)$ of (symmetric) **monoidal natural isomorphisms** between symmetric monoidal functors F and $G : \mathsf{SymMonGpd}(M, N)$ is defined as the Σ-type encoding a homotopy θ and families of 2-paths θ_0 and θ_2 as appearing in Definition 3.18. A **symmetric monoidal equivalence** $M \simeq N$ consists of two symmetric monoidal functors F and G in opposite directions, together with monoidal natural isomorphisms $\epsilon : \mathsf{SymMonFun}_{M,M}(G \circ F, \mathsf{id}_M)$ and $\eta : \mathsf{SymMonFun}_{N,N}(\mathsf{id}_N, F \circ G)$. Note that the symmetric aspect imposes no further conditions with respect to Definition 3.18.

Remark 4.9. Similarly to Lemma 3.20, an equivalence between the underlying types of two symmetric monoidal groupoids, where one of the half-adjoint functions is a symmetric monoidal functor, gives the other half-adjoint the structure of a symmetric monoidal functor.

The notions of functor from types to symmetric monoidal groupoids, and of freeness of those, are defined in the same way as those given in Section 3.3.

4.2 Symmetric Lists

In the proof of coherence for monoidal groupoids from Chapter 3, the terms in $\mathsf{list}(X)$ played the role of normal forms of monoidal expressions; indeed, in $\mathsf{FMG}(X)$ the distinction between e.g. terms $(a \otimes b) \otimes c$ and $a \otimes (b \otimes c)$ is ingrained in the definition of the constructor $\otimes$ (and reconciled by the constructor α), while associativity of the monoidal product in $\mathsf{list}(X)$ is simply a statement about a function which is defined *a posteriori*. Informally, the type $\mathsf{list}(X)$ should represent a free *strict* monoidal groupoid, even though strictness is not a property we can express (as discussed in Section 3.2). This can be seen from the fact that α_{list}, λ_{list} and ρ_{list} compute to the identity path when evaluated to *concrete* lists (see the discussion in Section 4.4), although this is not a statement that can be expressed *uniformly*, except for λ_{list}. Normalisation of monoidal expressions involves "forgetting" products $- \otimes e$ and $e \otimes -$, and a choice of directions of the brackets (for example, K and J in Definitions 3.49 and 3.50 make it so that the the brackets are right-leaning in normal forms).

For normal forms of symmetric monoidal expressions, we again require associativity and unitality to be "invisible", while symmetry will be related to groups of permutations. In this sense, we want to build a *permutative groupoid* from the type of lists, identifying those lists in the same orbit under the action of symmetric groups; that is, modulo homotopy, we want one path between any two lists for every distinct permutation that rearranges the elements of one into the other. We will use as

a reference the following standard (informal) presentation of the symmetric group S_n of order n with generators and relations: the groups S_0 and S_1 are trivial, while

$$\mathsf{S}_{n+2} \coloneqq \frac{(a_1, \ldots, a_{n+1})}{a_i^2 = 1, \quad a_{i+1}a_ia_{i+1} = a_ia_{i+1}a_i, \quad a_ia_j = a_ja_i \ \text{ for } |i-j| \geq 2.} \tag{4.10}$$

is the quotient of the free group on $\{a_1, \ldots, a_{n+1}\}$ by the specified relations.

Throughout this section, X is a type.

Definition 4.11 (Symmetric lists)**.** We define the type $\mathsf{slist}(X)$ of **symmetric** (unordered) **lists** as the following ap-recursive, 1-truncated HIT:[2]

$$\begin{aligned}
\mathsf{slist}(X) ::= \ & \mathsf{nil} : \mathsf{slist}(X) \\
| \ & \mathsf{cons} : X \to \mathsf{slist}(X) \to \mathsf{slist}(X) \quad \text{used with infix notation } :: \\
| \ & \mathsf{swap} : \Pi\,(x, y : X,\ l : \mathsf{slist}(X))\,.\, x :: y :: l = y :: x :: l \\
| \ & \mathsf{double} : \Pi\,(x, y : X,\ l : \mathsf{slist}(X))\,.\, \mathsf{swap}_{x,y,l} \cdot \mathsf{swap}_{y,x,l} = \mathsf{refl}_{x :: y :: l} \\
| \ & \mathsf{triple} : \Pi\,(x, y, z : X,\ l : \mathsf{slist}(X))\,.\,(\ldots) \\
| \ & T_{\mathsf{slist}} : \mathsf{IsHGpd}(\mathsf{slist}(X)),
\end{aligned}$$

where $\mathsf{triple}_{x,y,z,l}$ is the 2-path in Fig. 4.6.

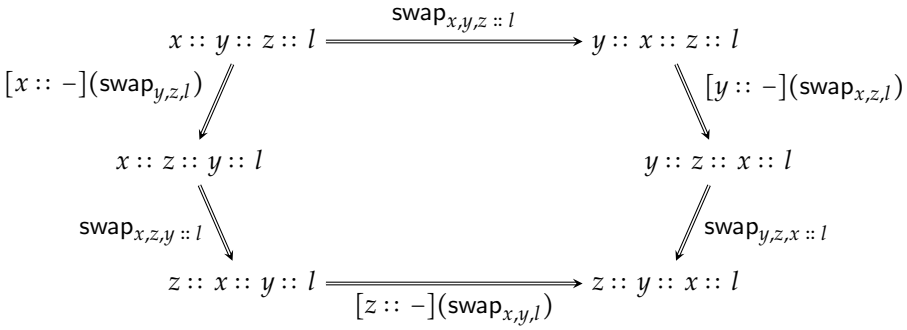


Figure 4.6: The constructor triple in the definition of $\mathsf{slist}(X)$.

In the definition above, we see a correspondence between the generator a_{n+1} in (4.10) and the 1-constructor swap of $\mathsf{slist}(X)$ (transposition of the first two elements in a list), while the other generators are related to the application $[x :: -]$ to a

[2] We thank S. Awodey for making us aware in June 2019 that the same HIT construction had been considered, in parallel, by V. Choudhury and M. Fiore [CF19]. Compare also with the construction of Kuratowski finite sets as listed finite sets in [Fru+18].

"swap", for some x (matching the inclusions $\mathsf{S}_n \hookrightarrow \mathsf{S}_{n+1}$). The first two relations in (4.10) correspond to the 2-constructors double and triple, while the last one is due to naturality of swap in its third argument [compare ML98, Chapter XI, proof of Theorem 1]. In this sense, informally, we have that the loop space of $\mathsf{slist}(\mathbf{1})$ at the (only) list of n element represents S_n, or better, the classifying space of all symmetric groups is represented by the type $\mathsf{slist}(\mathbf{1})$. We will investigate this claim in Chapter 5.

The constructor swap is natural in all arguments; as mentioned, we are interested in naturality in the list argument, which is stated in the following lemma.

Lemma 4.12. *For every $x, y : X$ and $p : l =_{\mathsf{slist}(X)} l'$, the diagram in Fig. 4.7 commutes.*

Proof. By induction on p. □

$$\begin{array}{ccc} x :: y :: l & \xrightarrow{\mathsf{swap}_{x,y,l}} & y :: x :: l \\ {\scriptstyle [x :: -]([y :: -](p))}\Big\downarrow & & \Big\downarrow{\scriptstyle [y :: -]([x :: -](p))} \\ x :: y :: l' & \xrightarrow[\mathsf{swap}_{y,x,l}]{} & y :: x :: l' \end{array}$$

Figure 4.7: Naturality of swap.

We argue that $\mathsf{slist}(X)$ is the carrier of a symmetric monoidal groupoid. In order to show this, we need to provide a monoidal product. Such a monoidal product is a "symmetric" version of list append, defined in the same way as $+\!\!+$ for lists on the 0-constructors of $\mathsf{slist}(X)$, so that it is compatible with the additional 1- and 2-constructors.

We remark that, from this point on, function extensionality needs to be assumed.

Definition 4.13 (Symmetric list append)**.** The function $- +\!\!+ - : \mathsf{slist}(X) \to \mathsf{slist}(X) \to \mathsf{slist}(X)$ is defined by induction (on its first argument):

- a term $\mathsf{nil}' : \mathsf{slist}(X) \to \mathsf{slist}(X)$ is given by $\mathsf{id}_{\mathsf{slist}(X)}$;
- given $x : X$ and a function $f : \mathsf{slist}(X) \to \mathsf{slist}(X)$, a term

$$\mathsf{cons}'(x, f) : \mathsf{slist}(X) \to \mathsf{slist}(X)$$

is given by $\mathsf{cons}(x) \circ f$;

- given $x, y : X$ and a function $f : \mathsf{slist}(X) \to \mathsf{slist}(X)$, a term

$$\mathsf{swap}'_{x,y,f} : \mathsf{cons}'(x, \mathsf{cons}'(y, f)) = \mathsf{cons}'(y, \mathsf{cons}'(x, f)) \tag{4.14}$$

  is obtained by function extensionality; indeed, the identity type in (4.14) is judgmentally equal to

$$\mathsf{cons}(x) \circ \mathsf{cons}(y) \circ f = \mathsf{cons}(y) \circ \mathsf{cons}(x) \circ f,$$

  which is inhabited by $\mathsf{fxt}(l \mapsto \mathsf{swap}_{x,y,f(l)})$;

- given $x, y : X$ and a function $f : \mathsf{slist}(X) \to \mathsf{slist}(X)$, a term

$$\mathsf{double}'_{x,y,f} : \mathsf{swap}'_{x,y,f} \cdot \mathsf{swap}'_{y,x,f} = \mathsf{refl}_{(l \mapsto \mathsf{cons}'(x,\mathsf{cons}'(y,f))(l))}$$

  is obtained along the following chain of identities:

$$\begin{aligned}
\mathsf{swap}'_{x,y,f} \cdot \mathsf{swap}'_{y,x,f} &\equiv \mathsf{fxt}(l \mapsto \mathsf{swap}_{x,y,f(l)}) \cdot \mathsf{fxt}(l \mapsto \mathsf{swap}_{y,x,f(l)}) \\
&= \mathsf{fxt}(l \mapsto \mathsf{swap}_{x,y,f(l)} \cdot \mathsf{swap}_{y,x,f(l)}) && \text{by Lemma 2.117} \\
&= \mathsf{fxt}(l \mapsto \mathsf{refl}_{x :: y :: f(l)}) && \text{by } [\mathsf{fxt}](\mathsf{fxt}(l \mapsto \mathsf{double}_{x,y,f(l)})) \\
&= \mathsf{refl}_{(l \mapsto x :: y :: f(l))} && \text{by Lemma 2.117} \\
&\equiv \mathsf{refl}_{(l \mapsto \mathsf{cons}'(x,\mathsf{cons}'(y,f))(l))};
\end{aligned}$$

- given $x, y, z : X$ and a function $f : \mathsf{slist}(X) \to \mathsf{slist}(X)$, a term

$$\begin{aligned}
\mathsf{triple}'_{x,y,z,f} : \mathsf{swap}'_{x,y,\mathsf{cons}'(z,f)} \cdot [\mathsf{cons}'(y)](\mathsf{swap}'_{x,z,f}) \cdot \mathsf{swap}'_{y,z,\mathsf{cons}'(x,f)} \\
= [\mathsf{cons}'(x)](\mathsf{swap}'_{y,z,f}) \cdot \mathsf{swap}'_{x,z,\mathsf{cons}'(y,f)} \cdot [\mathsf{cons}'(z)](\mathsf{swap}'_{x,y,f})
\end{aligned}$$

  is obtained first by noticing that there is a 2-path

$$[\mathsf{cons}'(a)](\mathsf{swap}'_{b,c,g}) = \mathsf{fxt}(l \mapsto [a :: -](\mathsf{swap}_{b,c,g(l)})),$$

  for every $a, b, c : X$ and $g : \mathsf{slist}(X) \to \mathsf{slist}(X)$ (by Lemma 2.118), and then using Lemma 2.117 and $[\mathsf{fxt}](\mathsf{fxt}(l \mapsto \mathsf{triple}_{x,y,z,f(l)}))$ similarly to the case for double'.

- the type of functions $\mathsf{slist}(X) \to \mathsf{slist}(X)$ is a 1-type, since the target type $\mathsf{slist}(X)$ is (Remark 2.75).

Some of the computation rules of $- +\!\!+ -$ state that, for every $x, y : X$ and $l : \mathsf{slist}(X)$:

$$(\mathsf{nil} +\!\!+ -) \equiv \mathsf{id}_{\mathsf{slist}(X)}; \tag{4.15}$$

$$((x :: l) +\!\!+ -) \equiv \mathsf{cons}(x) \circ (l +\!\!+ -); \tag{4.16}$$

$$[k \mapsto (k +\!\!+ -)](\mathsf{swap}_{x,y,l}) = \mathsf{fxt}(l' \mapsto \mathsf{swap}_{x,y,l +\!\!+ l'}); \tag{4.17}$$

the latter identity is between terms in the type of identities between functions:

$$(l' \mapsto x :: y :: l +\!\!+ l') =_{(\mathsf{slist}(X) \to \mathsf{slist}(X))} (l' \mapsto y :: x :: l +\!\!+ l').$$

Remark 4.18 (Notation). We use for $+\!\!+$ in $\mathsf{slist}(X)$ the same notational conventions as in $\mathsf{list}(X)$: the operation associates to the right, and we will omit parentheses in expressions such as $x :: l_1 +\!\!+ l_2$ (see Remark 2.35).

Lemma 4.19. *For every $x, y : X$ and $l_1, l_2 : \mathsf{slist}(X)$, there is a 2-path*

$$[- +\!\!+ l_2](\mathsf{swap}_{x,y,l_1}) = \mathsf{swap}_{x,y,l_1 +\!\!+ l_2}. \tag{4.20}$$

For every $x : X$ and paths $p : l_1 =_{\mathsf{slist}(X)} l_2$, $p : l_3 =_{\mathsf{slist}(X)} l_4$, there is a 2-path

$$[x :: -](p) +\!\!+ q = [x :: -](p +\!\!+ q). \tag{4.21}$$

Proof. We have:

$$\begin{aligned}
[- +\!\!+ l_2](\mathsf{swap}_{x,y,l_1}) &= [(-)(l_2)]([l \mapsto (l +\!\!+ -)](\mathsf{swap}_{x,y,l_1})) && \text{by path algebra} \\
&= [(-)(l_2)](\mathsf{fxt}(l' \mapsto \mathsf{swap}_{x,y,l_1 +\!\!+ l'})) && \text{via (4.17)} \\
&= \mathsf{swap}_{x,y,l_1 +\!\!+ l_2} && \text{by (2.115),}
\end{aligned}$$

proving (4.20) for every x, y, l_1 and l_2. A term in (4.21) is obtained by induction on p and q. □

We can now prove that the operation $+\!\!+$ is a symmetric monoidal product for $\mathsf{slist}(X)$.

Lemma 4.22. *The operation $+\!\!+$ is associative, i.e., we construct a term:*

$$\alpha_{\mathsf{slist}} : \Pi\,(l_1, l_2, l_3 : \mathsf{slist}(X))\,.\,(l_1 +\!\!+ l_2) +\!\!+ l_3 = l_1 +\!\!+ l_2 +\!\!+ l_3.$$

Proof. We proceed by induction on l_1, following the scheme for elimination in families of paths in a 1-type (which are 0-types):

- given $l_2, l_3 : \mathsf{slist}(X)$, we need a term

  $$\mathsf{nil}'(l_2, l_3) : (\mathsf{nil} +\!\!+ l_2) +\!\!+ l_3 = \mathsf{nil} +\!\!+ l_2 +\!\!+ l_3.$$

  Since both sides of the identity are judgmentally equal to $l_2 +\!\!+ l_3$, we can define $\mathsf{nil}'(l_2, l_3) :\equiv \mathsf{refl}_{l_2 +\!\!+ l_3}$;
- given $x : X$ and $l_1, l_2, l_3 : \mathsf{slist}(X)$, and assuming the inductive hypothesis

  $$h : (l_1 +\!\!+ l_2) +\!\!+ l_3 = l_1 +\!\!+ l_2 +\!\!+ l_3, \tag{4.23}$$

  we need a term

  $$\mathsf{cons}'(x, l_1, l_2, l_3, h) : \quad (x :: l_1 +\!\!+ l_2) +\!\!+ l_3 = x :: l_1 +\!\!+ l_2 +\!\!+ l_3.$$

As the left-hand side is judgmentally equal to $x :: (l_1 +\!\!+ l_2) +\!\!+ l_3$, we can define

$$\mathsf{cons}'(x, l_1, l_2, l_3, h) :\equiv [x :: -](h);$$

- given x, $y : X$, l_1, l_2, $l_3 : \mathsf{slist}(X)$ and an inductive hypothesis h as in (4.23), we need a term $\mathsf{swap}'(x, y, l_1, l_2, l_3, h)$ filling the outer diagram in Fig. 4.14; its construction is displayed in the same figure.

All other requirements are trivially fulfilled by the truncation level of the target types. The definition above makes α_{slist} compute as follows, for every $x : X$ and l_1, l_2, $l_3 : \mathsf{slist}(X)$:

$$\begin{aligned} \alpha_{\mathsf{slist}}(\mathsf{nil}, l_2, l_3) &\equiv \mathsf{refl}_{l_2 +\!\!+ l_3}, \\ \alpha_{\mathsf{slist}}(x :: l_1, l_2, l_3) &\equiv [x :: -](\alpha_{\mathsf{slist}}(l_1, l_2, l_3)). \end{aligned}$$

□

Lemma 4.24. *The operation $+\!\!+$ satisfies left and right unitality, i.e., we construct terms:*

$$\begin{aligned} \lambda_{\mathsf{slist}} &: \Pi\,(l : \mathsf{slist}(X))\,.\,\mathsf{nil} +\!\!+ l = l, \\ \rho_{\mathsf{slist}} &: \Pi\,(l : \mathsf{slist}(X))\,.\,l +\!\!+ \mathsf{nil} = l. \end{aligned}$$

Proof. Left unitality is trivial: since $\mathsf{nil} +\!\!+ l \equiv l$ for every $l : \mathsf{slist}(X)$, we can define $\lambda_{\mathsf{slist}}(l) :\equiv \mathsf{refl}_l$. For right unitality, we proceed by induction on l, similarly to the proof of associativity in Lemma 4.22:

- we need a term $\mathsf{nil}' : \mathsf{nil} +\!\!+ \mathsf{nil} = \mathsf{nil}$, which can be defined as $\mathsf{nil}' :\equiv \mathsf{refl}_{\mathsf{nil}}$;
- given $x : X$, $l : \mathsf{slist}(X)$ and assuming the inductive hypothesis $h : l +\!\!+ \mathsf{nil} = l$, we need a term
$$\mathsf{cons}'(x, l, h) : \quad x :: l +\!\!+ \mathsf{nil} = x :: l;$$
this can be defined as $\mathsf{cons}'(x, l, h) :\equiv [x :: -](h)$;
- given x, $y : X$, $l : \mathsf{slist}(X)$ and the inductive hypothesis $h : l +\!\!+ \mathsf{nil} = l$, we need a term $\mathsf{swap}'(x, y, l, h)$ filling the outer diagram in Fig. 4.15; its construction is displayed in the same figure.

The definition above makes ρ_{slist} compute as follows, for every $x : X$ and $l : \mathsf{slist}(X)$:

$$\begin{aligned} \rho_{\mathsf{slist}}(\mathsf{nil}) &\equiv \mathsf{refl}_{\mathsf{nil}}, \\ \rho_{\mathsf{slist}}(x :: l) &\equiv [x :: -](\rho_{\mathsf{slist}}(l)). \end{aligned}$$

□

Symmetry of $+\!\!+$ is less straightforward, and we will need a few preliminary results.

Lemma 4.25. *For every $x : X$ and $l_2 : \mathsf{slist}(X)$, there is a term*

$$Q_{x,l_2} : \Pi\,(l_1 : \mathsf{slist}(X))\,.\,x :: l_1 +\!\!+ l_2 = l_1 +\!\!+ x :: l_2. \tag{4.26}$$

Proof. By induction on l_1:

- we need a term $\mathsf{nil}'_{x,l_2} : x :: \mathsf{nil} +\!\!+ l_2 = \mathsf{nil} +\!\!+ x :: l_2$. Both sides of the identity are judgmentally equal to $x :: l_2$, so we can define $\mathsf{nil}' :\equiv \mathsf{refl}_{x :: l_2}$;
- given $y : X$, $l_1 : \mathsf{slist}(X)$ and the inductive hypothesis

$$h : \quad x :: l_1 +\!\!+ l_2 = l_1 +\!\!+ x :: l_2, \tag{4.27}$$

  we need a term

$$\mathsf{cons}'_{x,l_2}(y, l_1, h) : \quad x :: y :: l_1 +\!\!+ l_2 = y :: l_1 +\!\!+ x :: l_2.$$

  This can be given by the following concatenation:

$$\begin{aligned} x :: y :: l_1 +\!\!+ l_2 &= y :: x :: l_1 +\!\!+ l_2 && \text{by } \mathsf{swap}_{x,y,l_1 +\!\!+ l_2} \\ &= y :: l_1 +\!\!+ x :: l_2 && \text{by } [y :: -](h); \end{aligned}$$

- given $y, z : X$, $l_1 : \mathsf{slist}(X)$ and an inductive hypothesis h as in (4.27), we need a term $\mathsf{swap}'_{x,l_2}(y, z, l_1, h)$ filling the outer diagram in Fig. 4.16; the proof is displayed in the same figure.

The given definition makes Q_{x,l_2} compute as follows, for every $y : X$ and $l_2 : \mathsf{slist}(X)$:

$$\begin{aligned} Q_{x,l_2}(\mathsf{nil}) &\equiv \mathsf{refl}_{x :: l_2}, \\ Q_{x,l_2}(y :: l_1) &\equiv \mathsf{swap}_{x,y,l_1 +\!\!+ l_2} \cdot [y :: -](Q_{x,l_2}(l_1)). \end{aligned}$$

□

Lemma 4.28. *For every $x, y : X$ and $l_1 : \mathsf{slist}(X)$, there is a family of 2-paths $R_{x,y,l_1}(l_2)$, for $l_2 : \mathsf{slist}(X)$, filling the diagram in Fig. 4.8.*

Proof. We proceed by induction on l_2, following the scheme for elimination into a family of 2-paths in a groupoid (which are (-1)-types). We need:

- a 2-path nil'_{x,y,l_1}, corresponding to the diagram in Fig. 4.8 for $l_2 \equiv \mathsf{nil}$; this is trivially given, as the vertical paths compute to identity paths (unfolding the definition of Q), while the definition of $+\!\!+$ makes the path on the bottom judgmentally equal to $[\mathsf{id}_{\mathsf{slist}(X)}](\mathsf{swap}_{x,y,l_1})$, which is equal by path algebra to the path swap_{x,y,l_1} on the top;

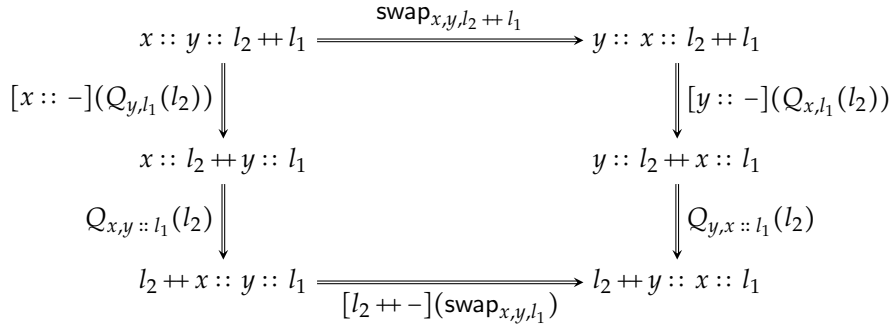


Figure 4.8: The 2-path $R_{x,y,l_1}(l_2)$.

- given $z : X$, $l_1 : \mathsf{slist}(X)$ and an inductive hypothesis h corresponding to the 2-path shown in Fig. 4.8, we need a term $\mathsf{cons}'_{x,y,l_1}(z, l_1, h)$ filling the outer diagram in Fig. 4.18; the proof is displayed in the same figure. □

Lemma 4.29. *The operation $+\!\!+$ is symmetric, i.e., we construct a term:*

$$\tau_{\mathsf{slist}} : \Pi\,(l_1, l_2 : \mathsf{slist}(X))\,.\, l_1 +\!\!+ l_2 = l_2 +\!\!+ l_1.$$

Proof. Finding such a term requires nested HIT-elimination (that is, induction on l_1 and, for each of the requirements, induction on l_2). The lemmata proved above help keeping this proof organized; induction on l_1 requires us to provide:

- a term $\mathsf{nil}' : \Pi\,(l_2 : \mathsf{slist}(X))\,.\, \mathsf{nil} +\!\!+ l_2 = l_2 +\!\!+ \mathsf{nil}$, which can be defined in the same way as ρ_{slist};
- given $x : X$, $l_1 : \mathsf{slist}(X)$ and assuming the inductive hypothesis

  $$h : \Pi\,(l_2 : \mathsf{slist}(X))\,.\, l_1 +\!\!+ l_2 = l_2 +\!\!+ l_1; \tag{4.30}$$

  a term

  $$\mathsf{cons}'(x, l_1, h) : \Pi\,(l_2 : \mathsf{slist}(X))\,.\, x :: l_1 +\!\!+ l_2 = l_2 +\!\!+ x :: l_1.$$

  This can be given, for every $l_2 : \mathsf{slist}(X)$, by the following chain of identities:

  $$\begin{aligned} x :: l_1 +\!\!+ l_2 &= x :: l_2 +\!\!+ l_1 && \text{by } [x :: -](h(l_2)) \\ &= l_2 +\!\!+ x :: l_1 && \text{by } Q_{x,l_1}(l_2), \end{aligned}$$

  with Q defined in Lemma 4.25;
- given $x, y : X$, $l_1 : \mathsf{slist}(X)$ and an inductive hypothesis h as in (4.30), a family of 2-paths $\mathsf{swap}'(x, y, l_1, h, l_2)$ filling the outer diagram in Fig. 4.17 for every $l_2 : \mathsf{slist}(X)$; the figure displays the construction, which makes use of the definition of R given in Lemma 4.28.

From the definition, τ_{slist} computes as follows, for $x, y : X$ and $l_1, l_2 : \mathsf{slist}(X)$:

$$\begin{aligned}
\tau_{\mathsf{slist}}(\mathsf{nil}, \mathsf{nil}) &\equiv \mathsf{refl}_{\mathsf{nil}}, \\
\tau_{\mathsf{slist}}(\mathsf{nil}, y :: l_2) &\equiv [y :: -](\tau_{\mathsf{slist}}(\mathsf{nil}, l_2)), \\
\tau_{\mathsf{slist}}(x :: l_1, \mathsf{nil}) &\equiv [x :: -](\tau_{\mathsf{slist}}(l_1, \mathsf{nil})) \cdot \mathsf{refl}_{x :: l_1}, \\
\tau_{\mathsf{slist}}(x :: l_1, y :: l_2) &\equiv [x :: -](\tau_{\mathsf{slist}}(l_1, y :: l_2)) \\
&\qquad \cdot \mathsf{swap}_{x,y,l_2 +\!\!+ l_1} \cdot [y :: -](Q_{x,l_1}(l_2)).
\end{aligned}$$

□

All is left to do is to verify the coherence diagrams.

Lemma 4.31. *Associativity and unitality of ++, defined as α_{slist}, λ_{slist} and ρ_{slist} in Lemmata 4.22 and 4.24, satisfy the coherence pentagon and triangle (Definition 3.11); these will be named $\pentagon_{\mathsf{slist}}$ and $\triangledown_{\mathsf{slist}}$.*

Proof. Both claims are proved by elimination on the leftmost list. The scheme for elimination into families of 2-paths in a 1-type only requires to provide 2-paths when the leftmost list is nil and when it is $x :: l$ for some $x : X$ and $l : \mathsf{slist}(X)$, provided recursively a 2-path for when it is l. The proof follows the one given in Remark 3.32 (Fig. 3.13) for list append. □

For the diagrams involving symmetry, we will make use, again, of a preliminary result.

Lemma 4.32. *For every $x : X$ and $l_1, l_3 : \mathsf{slist}(X)$, there is a family of 2-paths $H_{x,l_1,l_3}(l_2)$ for $l_2 : \mathsf{slist}(X)$, filling the diagram in Fig. 4.9, for α_{slist}, Q and τ_{slist} constructed in Lemmata 4.22, 4.25 and 4.29.*

Proof. We again use the elimination principle of $\mathsf{slist}(X)$ for families of 2-paths in a groupoid, where induction is performed on l_2:

- the derivation of a 2-path $\mathsf{nil}'_{x,l_1,l_3}$ is shown in Fig. 4.19a;
- the derivation of a 2-path $\mathsf{cons}'_{x,l_1,l_3}(y, l_2, h)$, for $y : X$, $l_2 : \mathsf{slist}(X)$ and the inductive hypothesis h corresponding to the 2-path shown in Fig. 4.9, is shown in Fig. 4.19b.

□

Lemma 4.33. *Associativity and symmetry of ++, defined as α_{slist} and τ_{slist} in Lemmata 4.22 and 4.29, satisfy the coherence hexagon (Definition 4.1), i.e., there is a term*

$$\begin{aligned}
\hexagon_{\mathsf{slist}} : \Pi\,(l_1, l_2, l_3 : \mathsf{slist}(X))\,. \\
\alpha_{\mathsf{slist}}(l_1, l_2, l_3) \cdot \tau_{\mathsf{slist}}(l_1, l_2 +\!\!+ l_3) \cdot \alpha_{\mathsf{slist}}(l_2, l_3, l_1) \\
= (\tau_{\mathsf{slist}}(l_1, l_2) +\!\!+ \mathsf{refl}_{l_3}) \cdot \alpha_{\mathsf{slist}}(l_2, l_1, l_3) \cdot (\mathsf{refl}_{l_2} +\!\!+ \tau_{\mathsf{slist}}(l_1, l_3)).
\end{aligned}$$

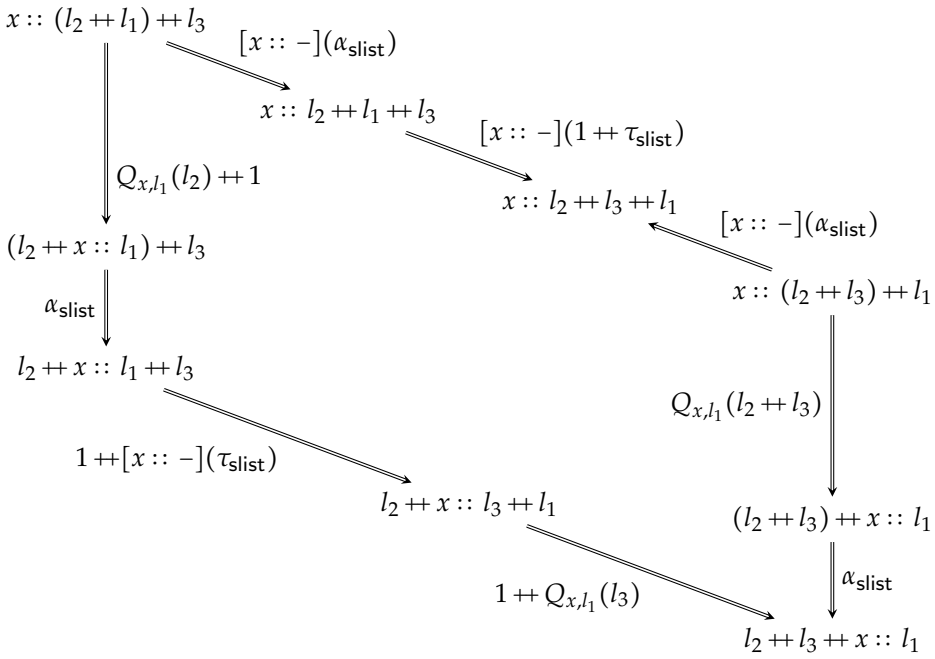


Figure 4.9: The 2-path $H_{x,l_1,l_3}(l_2)$.

Proof. We prove the claim by induction on l_1 and l_2, using nested instances of the elimination principle of $\mathsf{slist}(X)$, for which we only need to provide terms corresponding to the 0-constructors nil and cons:

- the 2-path $\circlearrowleft_{\mathsf{slist}}(\mathsf{nil}, \mathsf{nil}, l_3)$ is shown in Fig. 4.20a for every $l_3 : \mathsf{slist}(X)$;
- given $y : X$, l_2 and, as inductive hypothesis, $\circlearrowleft_{\mathsf{slist}}(\mathsf{nil}, l_2, l_3)$ for every $l_3 : \mathsf{slist}(X)$, the 2-path $\circlearrowleft_{\mathsf{slist}}(\mathsf{nil}, y :: l_2, l_3)$ is derived for every $l_3 : \mathsf{slist}(X)$ as in Fig. 4.20b;
- given $x : X$, $l_1 : \mathsf{slist}(X)$ and, as inductive hypothesis, $\circlearrowleft_{\mathsf{slist}}(l_1, l_2, l_3)$ for every l_2, $l_3 : \mathsf{slist}(X)$, the 2-path $\circlearrowleft_{\mathsf{slist}}(x :: l_1, l_2, l_3)$ is derived for every $l_3 : \mathsf{slist}(X)$ as in Fig. 4.20c (which uses the 2-path of H defined in Lemma 4.32 by induction on l_2). □

Lemma 4.34. *Symmetry of ++, defined as τ_{slist} in Lemma 4.29, satisfies the coherence bigon (Definition 4.1), i.e., there is a term*

$$\circlearrowleft_{\mathsf{slist}} : \Pi\,(l_1, l_2 : \mathsf{slist}(X))\,.\,\tau_{\mathsf{slist}}(l_1, l_2) \cdot \tau_{\mathsf{slist}}(l_2, l_1) = \mathsf{refl}_{l_1 ++ l_2}.$$

Proof. Once again, we prove the claim by induction on l_1 and l_2, using nested instances of the elimination principle of $\mathsf{slist}(X)$, for which we only need to provide terms corresponding to the 0-constructors (nil and cons):

- the 2-path $\hexagon_{\mathsf{slist}}(\mathsf{nil}, \mathsf{nil})$ is trivial, as $\tau_{\mathsf{slist}}(\mathsf{nil}, \mathsf{nil}) \equiv \mathsf{refl}_{\mathsf{nil}}$;
- given $y : X$, $l_2 : \mathsf{slist}(X)$ and assuming as inductive hypothesis a term

$$\hexagon_{\mathsf{slist}}(\mathsf{nil}, l_2) : \tau_{\mathsf{slist}}(\mathsf{nil}, l_2) \cdot \tau_{\mathsf{slist}}(l_2, \mathsf{nil}) = \mathsf{refl}_{l_2},$$

  a term $\hexagon_{\mathsf{slist}}(\mathsf{nil}, y :: l_2)$ can be obtained from $[[y :: -]](\hexagon_{\mathsf{slist}}(\mathsf{nil}, l_2))$, since $\tau_{\mathsf{slist}}(\mathsf{nil}, y :: l_2) \equiv [y :: -](\tau_{\mathsf{slist}}(\mathsf{nil}, l_2))$ and $\tau_{\mathsf{slist}}(y :: l_2, \mathsf{nil}) \equiv [y :: -](l_2, \mathsf{nil}) \cdot \mathsf{refl}_{y :: l}$;
- given $x : X$, $l_1 : \mathsf{slist}(X)$, and assuming as inductive hypothesis a term

$$\hexagon_{\mathsf{slist}}(l_1, l_2) : \tau_{\mathsf{slist}}(l_1, l_2) \cdot \tau_{\mathsf{slist}}(l_2, l_1) = \mathsf{refl}_{l_1 +\!\!+ l_2} \tag{4.35}$$

  for every $l_2 : \mathsf{slist}(X)$, a term $\hexagon_{\mathsf{slist}}(x :: l_1, \mathsf{nil})$ can be obtained, similarly to the previous case, from $[[x :: -]](\hexagon_{\mathsf{slist}}(l_1, \mathsf{nil}))$;
- given $x, y : X$, $l_1, l_2 : \mathsf{slist}(X)$, and assuming the inductive hypothesis as in (4.35) and a further inductive hypothesis

$$\hexagon_{\mathsf{slist}}(x :: l_1, l_2) : \tau_{\mathsf{slist}}(x :: l_1, l_2) \cdot \tau_{\mathsf{slist}}(l_2, x :: l_1) = \mathsf{refl}_{x :: l_1 +\!\!+ l_2},$$

  a term $\hexagon_{\mathsf{slist}}(x :: l_1, y :: l_2)$ can be obtained as shown in Fig. 4.21. □

Corollary 4.36. *The operation $+\!\!+$ is a symmetric monoidal product for* $\mathsf{slist}(X)$.

Proof. Follows from the lemmata in this section. □

4.3 Coherence for Symmetric Monoidal Groupoids

Similarly to the construction of $\mathsf{FMG}(X)$ in Chapter 3, we will define, for a type X, a free symmetric monoidal groupoid $\mathsf{FSMG}(X)$ as a HIT with constructors typed according to the definition of a free symmetric monoidal structure. Coherence for symmetric monoidal groupoids will entail showing that, for every 0-type X, there is a symmetric monoidal equivalence

$$\mathsf{FSMG}(X) \simeq \mathsf{slist}(X) \tag{4.37}$$

between $\mathsf{FSMG}(X)$ and the "simpler" symmetric monoidal groupoid $\mathsf{slist}(X)$, which does not impose associativity and unitality of its product as constructors (these have been proved in Lemma 4.22 and Lemma 4.24 instead).

Definition 4.38 (FSMG)**.** Given a type $X : \mathcal{U}$, we define the ap-recursive, 1-truncated HIT $\mathsf{FSMG}(X)$ with the following constructors:

$$\begin{aligned}\mathsf{FSMG}(X) ::= {} & e : \mathsf{FSMG}(X) \mid \iota : X \to \mathsf{FSMG}(X) \\ & \mid \otimes : \mathsf{FSMG}(X) \to \mathsf{FSMG}(X) \to \mathsf{FSMG}(X) \\ & \mid \alpha : \Pi\,(a, b, c : \mathsf{FSMG}(X))\,.\,(a \otimes b) \otimes c = a \otimes (b \otimes c) \\ & \mid \lambda : \Pi\,(b : \mathsf{FSMG}(X))\,.\,e \otimes b = b \mid \rho : \Pi\,(a : \mathsf{FSMG}(X))\,.\,a \otimes e = a \\ & \mid \tau : \Pi\,(a, b : \mathsf{FSMG}(X))\,.\,a \otimes b = b \otimes a \\ & \mid \pentagon : \ldots \mid \triangledown : \ldots \mid \hexagon : \ldots \mid \diamond : \ldots \\ & \mid T : \mathsf{IsHGpd}(\mathsf{FSMG}(X)),\end{aligned}$$

where $\pentagon$, $\triangledown$, $\hexagon$ and $\diamond$ are families of 2-path constructors corresponding to the coherence diagrams of a symmetric monoidal groupoid. The elimination principle follows the scheme given in Section 2.6.

The construction of FSMG is functorial: the function

$$\mathsf{FSMG} :\equiv (X \mapsto \langle\langle \mathsf{FSMG}(X), T\rangle, e, \otimes, \alpha, \lambda, \rho, \tau, \pentagon, \triangledown, \hexagon, \diamond\rangle) \tag{4.39}$$

is a functor, as seen for FMG in Lemma 3.38. Moreover, $\mathsf{FSMG}(X)$ is freely generated by X.

Lemma 4.40. *The functor* FSMG *in (4.39) is free.*

Proof. The proof follows closely that of Corollary 3.47. □

The construction of the equivalence in (4.37) will result in the normalisation of symmetric monoidal expressions into (symmetric) list expressions. As the underlying higher inductive types $\mathsf{FSMG}(X)$ and $\mathsf{slist}(X)$ are built upon the same constructors as those of $\mathsf{FMG}(X)$ and $\mathsf{list}(X)$, with additional constructors allowing the respective monoidal structures to be symmetric, normalisation of symmetric monoidal expressions will agree with normalisation of monoidal expressions in all definition and proofs involving the common constructors. Informally, this can be interpreted as the fact that the "strictification" of associativity and unitality in a monoidal groupoid is *compatible* with the addition of an auxiliary symmetric structure, and thus the diagram in Fig. 4.10 will commute.

Throughout the rest of this section, X is a 0-type.

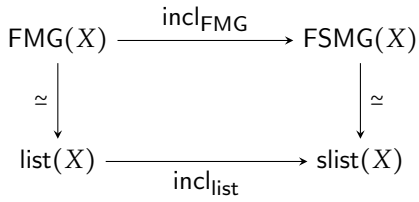


Figure 4.10: The inclusions $\mathsf{incl}_{\mathsf{FMG}} : \mathsf{FMG}(X) \to \mathsf{FSMG}(X)$ and $\mathsf{incl}_{\mathsf{list}} : \mathsf{list}(X) \to \mathsf{slist}(X)$, as monoidal functors, are compatible with the equivalences determining coherence for monoidal and symmetric monoidal groupoids: the diagram above commutes. The equivalence on the left is given in Corollary 3.56; the one on the right will be constructed in this section (Corollary 4.49).

Definition 4.41. We define a symmetric monoidal functor

$$K : \mathsf{SymMonGpd}(\mathsf{FSMG}(X), \mathsf{slist}(X)).$$

The definition goes along the construction of the monoidal functor K in the previous chapter (Definition 3.49). The underlying function $K : \mathsf{FSMG}(X) \to \mathsf{slist}(X)$ between groupoids can be given by the elimination principle of $\mathsf{FSMG}(X)$, sending the monoidal structure of $\mathsf{FSMG}(X)$ to the one of $\mathsf{slist}(X)$, described in the previous section, and $\iota(x)$ to $x :: \mathsf{nil}$. The paths $K_0 : \mathsf{nil} = K(e)$ and $K_2(a, b) : K(a) +\!\!+ K(b) = K(a \otimes b)$ for $a, b : \mathsf{FSMG}(X)$ are identity paths; the computation rules of K provide 2-paths K_α, K_λ, K_ρ and K_τ witnessing the following identities:

$$\alpha_{\mathsf{slist}} = [K](\alpha), \qquad \lambda_{\mathsf{slist}} = [K](\lambda), \qquad \rho_{\mathsf{slist}} = [K](\rho), \qquad \tau_{\mathsf{slist}} = [K](\tau). \tag{4.42}$$

Definition 4.43. We define a symmetric monoidal functor

$$J : \mathsf{SymMonGpd}(\mathsf{slist}(X), \mathsf{FSMG}(X)).$$

The underlying function $J : \mathsf{slist}(X) \to \mathsf{FSMG}(X)$ is defined using the elimination principle of $\mathsf{slist}(X)$; on 0-constructors, its behaviour will be akin to the one of the monoidal functor J in the previous chapter. We need:

- a term $\mathsf{nil}' : \mathsf{FSMG}(X)$, which will be the image of nil; this is given by e;
- for every $x : X$ and $a : \mathsf{FSMG}(X)$, a term $\mathsf{cons}'(x, a)$, given by $\iota(x) \otimes a$;
- for every $x, y : X$ and $a : \mathsf{FSMG}(X)$, a term $\mathsf{swap}'_{x,y,a} : \mathsf{cons}'(x, \mathsf{cons}'(y, a)) = \mathsf{cons}'(y, \mathsf{cons}'(x, a))$; this is obtained along the following chain of identities:

$$\begin{aligned} \mathsf{cons}'(x, \mathsf{cons}'(y, a)) &\equiv \iota(x) \otimes (\iota(y) \otimes a) \\ &= (\iota(x) \otimes \iota(y)) \otimes a && \text{by } \alpha^{-1} \end{aligned}$$

$$\begin{aligned} &= (\iota(y) \otimes \iota(x)) \otimes a && \text{by } \tau \otimes \mathsf{refl}_a \\ &= \iota(y) \otimes (\iota(x) \otimes a) && \text{by } \alpha \\ &\equiv \mathsf{cons}'(y, \mathsf{cons}'(x, a)); \end{aligned}$$

- for every $x, y : X$ and $a : \mathsf{FSMG}(X)$, a 2-path $\mathsf{double}'_{x,y,a}$ given as the outer diagram in Fig. 4.22;
- for every $x, y, z : X$ and $a : \mathsf{FSMG}(X)$, a 2-path $\mathsf{triple}'_{x,y,z,a}$ given as the outer diagram in Fig. 4.23.

The requirement about $\mathsf{FSMG}(X)$ being a 1-type is fulfilled by the constructor T. The computation rules of J state:

$$\begin{aligned} J(\mathsf{nil}) &\equiv e, \\ J(x :: l) &\equiv \iota(x) \otimes J(l), \\ [J](\mathsf{swap}_{x,y,l}) &= \alpha^{-1} \cdot (\tau \otimes \mathsf{refl}) \cdot \alpha, \end{aligned} \tag{4.44}$$

for every $x, y : X$ and $l : \mathsf{slist}(X)$.

The remaining data for a symmetric monoidal functor is then given by "completing" the definition given for the analogue in Definition 3.50. A path $J_0 : e = J(\mathsf{nil})$ is defined to be the identity path. A family of paths

$$J_2 : \Pi\,(l_1, l_2 : \mathsf{slist}(X))\,.\, J(l_1) \otimes J(l_2) = J(l_1 +\!\!+ l_2)$$

is defined by induction on l_1, where the requirements for 0-constructors are provided in the same way as in Definition 3.50:

- a term $\mathsf{nil}'(l_2) : J(\mathsf{nil}) \otimes J(l_2) = J(\mathsf{nil} +\!\!+ l_2)$ for every $l_2 : \mathsf{slist}(X)$ is given by $\lambda_{J(l_2)}$, as the left-hand side computes to $e \otimes J(l_2)$ and the right-hand side computes to $J(l_2)$;
- for $x : X$, $l_1 : \mathsf{slist}(X)$ and the inductive hypothesis

  $$h : \Pi\,(l_2 : \mathsf{slist}(X))\,.\, J(l_1) \otimes J(l_2) = J(l_1 +\!\!+ l_2), \tag{4.45}$$

  a term

  $$\mathsf{cons}'(x, l_1, h, l_2) : J(x :: l_1) \otimes J(l_2) = J(x :: l_1 +\!\!+ l_2)$$

  is constructed for every $l_2 : \mathsf{slist}(X)$, once unfolding the definition of J, by

  $$\alpha \cdot (\mathsf{refl}_{\iota(x)} \otimes h(l_2));$$

- given $x, y : X$, $l_1 : \mathsf{slist}(X)$ and an induction hypothesis h as in (4.45), a 2-path $\mathsf{swap}'(x, y, l_1, h, l_2)$ for every $l_2 : \mathsf{slist}(X)$, filling the outer diagram in Fig. 4.24 is provided as shown in the figure.

The 2-paths J_α, J_λ and J_ρ are obtained exactly as in Definition 3.50, since the elimination principle of $\mathsf{slist}(X)$ to families of 2-paths in groupoids only concerns 2-paths corresponding to the 0-constructors nil and cons, and J_0 and J_2 compute in the same way on terms constructed in such a way. Finally, a 2-path $J_\tau(l_1, l_2)$ corresponding to the diagram in Fig. 4.11 for every l_1, $l_2 : \mathsf{slist}(X)$ is given by induction on l_1 and l_2:

- $J_\tau(\mathsf{nil}, \mathsf{nil})$ is displayed in Fig. 4.25a;
- $J_\tau(\mathsf{nil}, y :: l_2)$, for $y : X$ and $l_2 : \mathsf{slist}(X)$ and assuming $J_\tau(\mathsf{nil}, l_2)$ given, is displayed in Fig. 4.25b;
- for $x : X$, l_1, $l_2 : \mathsf{slist}(X)$ and assuming $J_\tau(l_1, l_2)$ given, the construction of the 2-path $J_\tau(x :: l_1, l_2)$ is displayed in Fig. 4.25c. Induction on l_2 affects only one part of the diagram; this is signalled in the same figure as a 2-path $V_{x,l_1}(l_2)$, which is filled by induction on l_2 in the ways depicted in Fig. 4.26a and Fig. 4.26b (respectively, for $V_{x,l_1}(\mathsf{nil})$ and $V_{x,l_1}(y :: l_2)$, for every $y : X$).

$$\begin{array}{ccc} J(l_1) \otimes J(l_2) & \xrightarrow{\quad\tau\quad} & J(l_2) \otimes J(l_1) \\ {\scriptstyle J_2(l_1,l_2)}\Big\downarrow & & \Big\downarrow{\scriptstyle J_2(l_2,l_1)} \\ J(l_1 +\!\!+ l_2) & \xrightarrow[[J](\tau_{\mathsf{slist}})]{} & J(l_2 +\!\!+ l_1) \end{array}$$

Figure 4.11: The 2-path J_τ.

Lemma 4.46. *There is a symmetric monoidal natural isomorphism*

$$\eta : \mathsf{SymMonFun}_{\mathsf{FSMG}(X),\mathsf{FSMG}(X)}(\mathsf{id}, J \circ K).$$

Proof. The proof is identical to that of Lemma 3.52, save for the missing requirement τ' in the definition of the underlying homotopy $\eta : \mathsf{id} \sim J \circ K$; this is illustrated in Fig. 4.27. □

Lemma 4.47. *There is a symmetric monoidal natural isomorphism*

$$\epsilon : \mathsf{SymMonFun}_{\mathsf{slist}(X),\mathsf{slist}(X)}(K \circ J, \mathsf{id}).$$

Proof. Once again, we can adapt the proof given for $\mathsf{FMG}(X)$ and $\mathsf{list}(X)$ in the previous chapter (Lemma 3.54). The underlying homotopy $\epsilon : K \circ J \sim \mathsf{id}$ can be obtained by the elimination principle of $\mathsf{slist}(X)$, where:

- a path $\mathsf{nil}' : K(J(\mathsf{nil})) = \mathsf{nil}$ is given by the identity path;

- for $x : X$, $l : \mathsf{slist}(X)$ and an inductive hypothesis $h : K(J(l)) = l$, a path

$$\mathsf{cons}'_{x,l,h} : K(J(x :: l)) = x :: l$$

is given by $[x :: -](h)$, since the left-hand side of the identity computes to $x :: K(J(l))$;

- for $x, y : X$, $l : \mathsf{slist}(X)$ and an inductive hypothesis h as above, a 2-path $\mathsf{swap}'_{x,y,l,h}$ corresponding to the diagram in Fig. 4.28 is provided in the figure itself, using the following chain of identities:

$$\begin{aligned}
&[K \circ J](\mathsf{swap}_{x,y,l}) = [K]([J](\mathsf{swap}_{x,y,l})) \\
&\quad = [K](\alpha^{-1} \cdot (\tau \otimes \mathsf{refl}_{J(l)}) \cdot \alpha) && \text{by (4.44)} \\
&\quad = ([K](\alpha))^{-1} \cdot [K](\tau \otimes \mathsf{refl}_{J(l)}) \cdot [K](\alpha) \\
&\quad = \alpha^{-1}_{\mathsf{slist}} \cdot [K](\tau \otimes \mathsf{refl}_{J(l)}) \cdot \alpha_{\mathsf{slist}} && \text{comp. rule of } K \\
&\quad \equiv \mathsf{refl}_{x :: y :: J(l)} \cdot [K](\tau \otimes \mathsf{refl}_{J(l)}) \cdot \mathsf{refl}_{y :: x :: J(l)} \\
&\quad = [K](\tau \otimes \mathsf{refl}_{J(l)}) = [K](\tau) +\!\!+ \mathsf{refl}_{K(J(k))} \\
&\quad = [- +\!\!+ K(J(l))]([K](\tau)) = [- +\!\!+ K(J(l))](\tau_{\mathsf{slist}}) && \text{comp. rule of } J \\
&\quad \equiv [- +\!\!+ K(J(l))](\mathsf{refl}_{x :: y :: \mathsf{nil}} \cdot \mathsf{swap}_{x,y,\mathsf{nil}} \cdot \mathsf{refl}_{y :: x :: \mathsf{nil}}) \\
&\quad = [- +\!\!+ K(J(l))](\mathsf{swap}_{x,y,\mathsf{nil}}) = \mathsf{swap}_{x,y,\mathsf{nil} +\!\!+ K(J(l))} && \text{by (4.20)} \\
&\quad \equiv \mathsf{swap}_{x,y,K(J(l))}. && (4.48)
\end{aligned}$$

The diagrams ϵ_0 and ϵ_2 are as in Lemma 3.54. □

Corollary 4.49 (Coherence for symmetric monoidal groupoids)**.** *There is a symmetric monoidal equivalence* $\mathsf{FSMG}(X) \simeq \mathsf{slist}(X)$.

Proof. Follows from the lemmata in this section. □

4.4 Discussion

Before proceeding to Chapter 5 with a further analysis of free symmetric monoidal groupoids, we summarize the results achieved with Corollary 4.49:

- we presented a description of the HIT $\mathsf{FSMG}(X)$ of free symmetric monoidal expressions in terms of a simpler type $\mathsf{slist}(X)$ of "lists with added paths", with considerably fewer constructors. The equivalence $\mathsf{FSMG}(X) \simeq \mathsf{slist}(X)$ is accomplished by a process of normalisation, which is done by means of a function preserving the symmetric monoidal structure (a symmetric monoidal functor);

- conversely, we showed that the families of constructors that we added to $\mathsf{list}(X)$ to define the type $\mathsf{slist}(X)$ (namely, swap, double, triple and T_{slist} of $\mathsf{slist}(X)$) are sufficient to make the latter a free symmetric monoidal groupoid;
- moreover, the symmetric monoidal equivalence $\mathsf{FSMG}(X) \simeq \mathsf{slist}(X)$ shows that the presence of symmetry axioms in $\mathsf{FSMG}(X)$ – as additional constructors to $\mathsf{FMG}(X)$ – does not impair the possibility of normalising associativity (i.e. forcing a right-leaning bracketing for expressions) and unitality (cancelling of the unit), while retaining symmetry; the example in Fig. 3.8 for $\mathsf{FMG}(X)$ then still applies when symmetry is present.

We reiterate that, while function extensionality was not needed in order to prove coherence for monoidal groupoids, we had to assume it for coherence for symmetric monoidal groupoids. Indeed, the operation $+\!+ : \mathsf{slist}(X) \to \mathsf{slist}(X) \to \mathsf{slist}(X)$ is defined by means of the elimination principle of $\mathsf{slist}(X)$, which asks, because of its 1-constructor swap, to provide a path between two *functions* in $\mathsf{slist}(X) \to \mathsf{slist}(X)$ (Definition 4.13).

An important meta-theoretical observation is that, while α_{slist}, λ_{slist} and ρ_{slist} all compute to reflexivity paths when instantiated to *explicit* terms (as their counterparts for $\mathsf{list}(X)$ do), τ_{slist} obviously does not. This is due to the fact that $\mathsf{slist}(X)$ is *not* a 0-type; the normal form of a symmetric monoidal expression can only be given up to permutation of elements, which mirrors the symmetry of the product. This agrees with the classical formulation of coherence for symmetric monoidal categories, which states that it is possible to strictify associativity and unitality, but not symmetry.

The observation above is not immediately evident when the type X of generators is a (-1)-type: the monoidal product $+\!+$ in lists is not symmetric, but, as mentioned in Section 3.7, $\mathsf{list}(\mathbf{1})$ is equivalent to $\mathbb{N}$, whose monoidal product (addition of natural numbers) happens to be symmetric. Indeed, it is easy to produce an equivalence

$$\|\mathsf{slist}(\mathbf{1})\|_0 \simeq \mathbb{N},$$

showing that the "permutative" character of $\mathsf{slist}(\mathbf{1})$ is not detected at a set level. Indeed, one could prove that $\mathsf{slist}(\mathbf{1})$ is *not* equivalent to $\mathbb{N}$: this can be done by showing that $\mathsf{slist}(\mathbf{1})$ is not a 0-type, while we saw in Lemma 3.27 that $\mathsf{list}(\mathbf{1})$ is a 0-type.

The proof that $\mathsf{slist}(\mathbf{1})$ is not a 0-type is given, indirectly, in the next chapter, as a consequence of Theorem 5.35 and Lemma 5.46. Such a proof uses univalence and is similar to the one presented in [Uni13, Lemma 6.4.1] to show that S^1 is not a 0-type.

Roughly, the argument is as follows: a function

$$f : \mathsf{slist}(\mathbf{1}) \to \mathcal{U}$$

is defined in some way, so that $f(* :: * :: \mathsf{nil}) \equiv \mathbf{2}$ and such that there is a path

$$p : [f](\mathsf{swap}_{*,*,\mathsf{nil}}) =_{(\mathbf{2}=\mathbf{2})} \mathsf{ua}(\omega),$$

where $\omega : \mathbf{2} \simeq \mathbf{2}$ is the non-identity equivalence, so $(\omega = \mathsf{id}_{\mathbf{2}}) \to \mathbf{0}$. Since we also have a path

$$p' : [f](\mathsf{refl}_{* :: * :: \mathsf{nil}}) =_{(\mathbf{2}=\mathbf{2})} \mathsf{ua}(\mathsf{id}_{\mathbf{2}}),$$

if we assume that $\mathsf{slist}(\mathbf{1})$ is a 0-type, we obtain a path $r : \mathsf{swap}_{*,*,\mathsf{nil}} = \mathsf{refl}_{* :: * :: \mathsf{nil}}$, and therefore a proof that $\omega = \mathsf{id}_{\mathbf{2}}$, which results in a contradiction. This matches our understanding that $\mathsf{swap}_{*,*,\mathsf{nil}}$ ought *not* to be identified with the trivial path.

There is a complication in the argument above with respect to the proof in [Uni13, Lemma 6.4.1] for S^1. The type S^1 is a 1-HIT with one 0-constructor $(\mathsf{base} : \mathsf{S}^1)$ and one 1-constructor $(\mathsf{loop} : \mathsf{base} = \mathsf{base})$, so a function $f : \mathsf{S}^1 \to \mathcal{U}$ making the argument work can be defined rather easily by the elimination principle of S^1. In contrast, $\mathsf{slist}(\mathbf{1})$ is a 1-truncated 2-HIT. This means that not only we need to make sure that $[f](\mathsf{swap}_{*,*,\mathsf{nil}})$ is "compatible" with the higher constructors of $\mathsf{slist}(\mathbf{1})$, but also – crucially – we cannot eliminate directly into $\mathcal{U}$, because $\mathcal{U}$ is not a 1-type. A solution is to eliminate into a subuniverse $\Sigma\,(X : \mathcal{U})\,.\,P(X)$ which is provably a 1-type, and then take the first projection $\mathsf{pr}_1 : (\Sigma\,(X : \mathcal{U})\,.\,P(X)) \to \mathcal{U}$.

We will discuss the truncation level of a type equivalent to $\mathsf{slist}(\mathbf{1})$ in the next chapter, where we will be concerned with establishing a relationship between free symmetric monoidal groupoids and the classifying spaces of symmetric groups.

4.5 Figures in Proofs

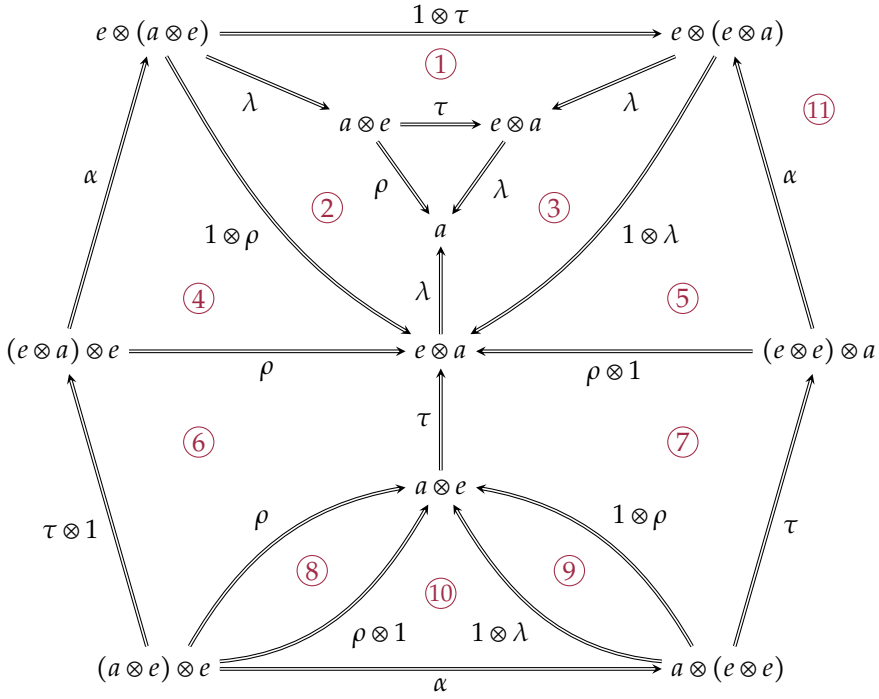


Figure 4.12: Commutativity of the diagram in Fig. 4.4a, here appearing as the unmarked triangle; the $-_M$ is omitted from terms for readability. The 2-paths (1), (2) and (3) are instances of naturality of λ_M; (4), (8) and (9) are instances of the diagrams in Fig. 3.4b, Fig. 3.4e and Fig. 3.4c respectively; (5) and (10) are instances of $\triangledown_M$; (6) and (7) are instances of naturality of ρ_M and τ_M respectively; the outer hexagon (11) is an instance of $\hexagon_M$.

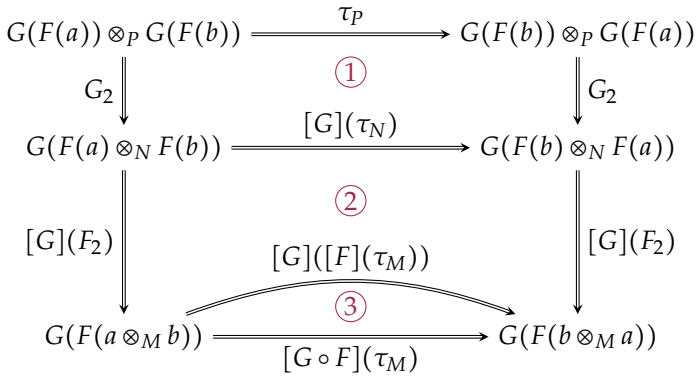


Figure 4.13: Derivation of the 2-path $(G \circ F)_\tau$, after unfolding the definition of $(G \circ F)_2$. The 2-paths (1) and (2) follow from G_τ and $[[G]](F_\tau)$ respectively; (3) is given by path algebra.

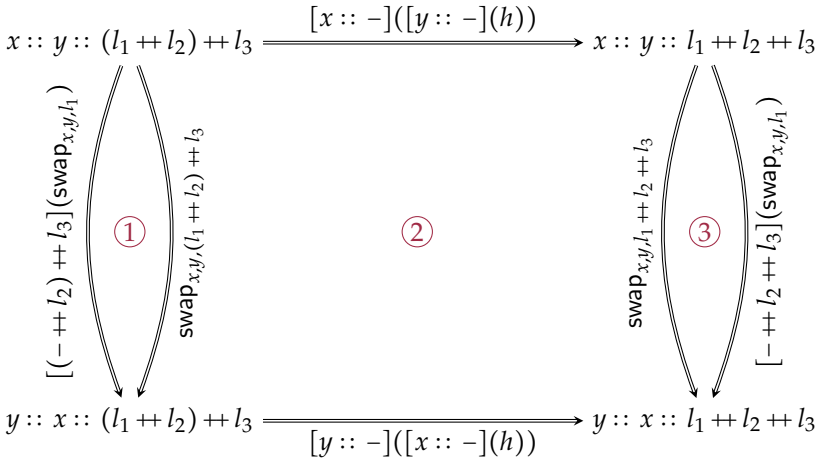


Figure 4.14: The term $\mathsf{swap}'(x, y, l_1, l_2, l_3, h)$ in the inductive definition of α_{slist}, after unfolding the definition of cons'. The 2-paths in (1) and (3) are obtained via (4.20); (2) is an instance of naturality of swap.

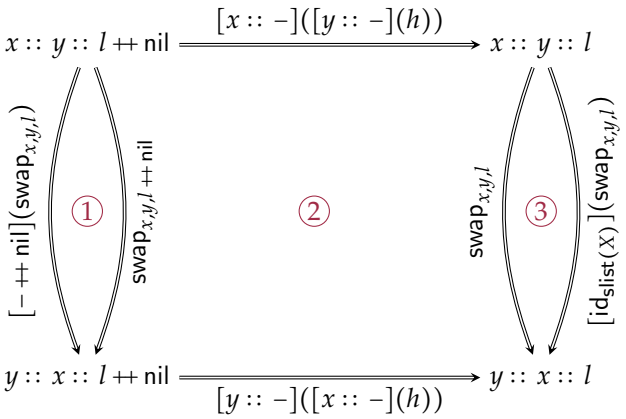


Figure 4.15: The term $\mathsf{swap}'(x, y, l, h)$ in the inductive definition of ρ_{slist}, after unfolding the definition of cons'. The 2-path in (1) is given by path algebra; (2) is an instance of naturality of swap; (3) is obtained via (4.20).

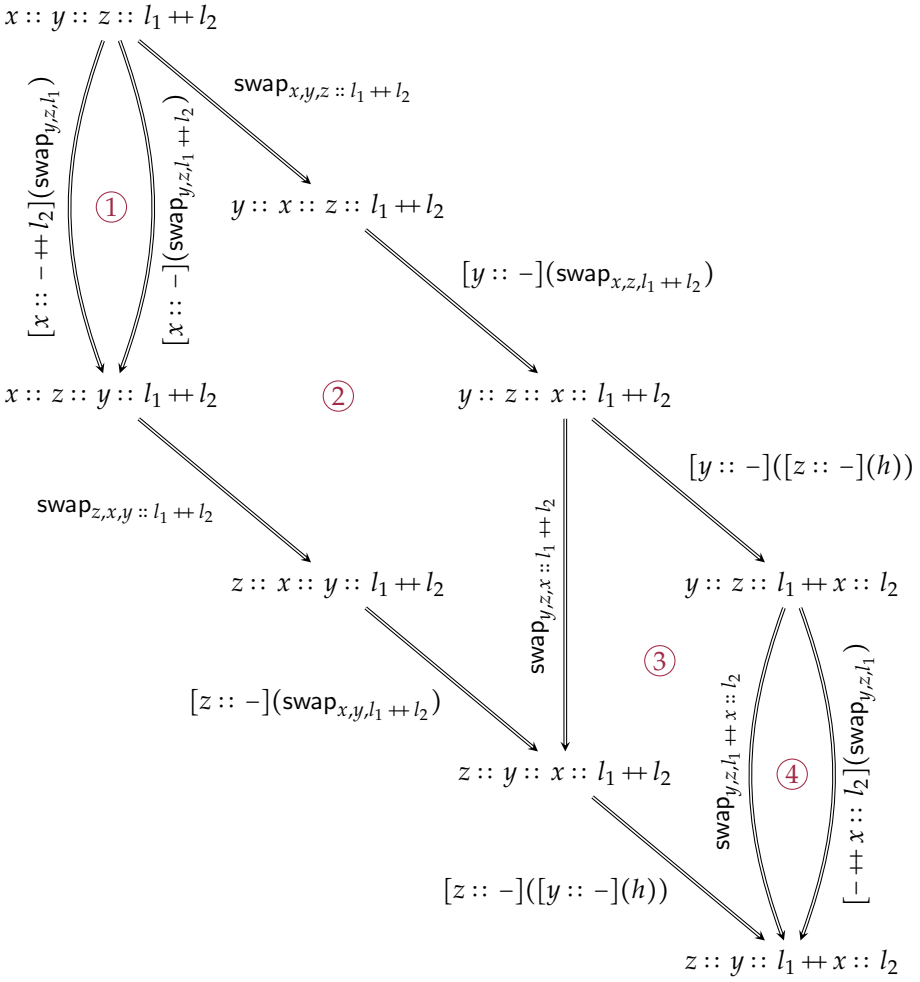


Figure 4.16: The term $\mathsf{swap}'_{x,l_2}(y, z, l_1, h)$ in the inductive definition of Q_{x,l_2}, after unfolding the definition of cons'. The 2-paths in (1) and (4) are filled by (4.20) and path algebra (for (1) we use $[x :: - +\!\!+ l_2](\mathsf{swap}_{y,z,l_1}) = [x :: -]([- +\!\!+ l_2](\mathsf{swap}_{y,z,l_1}))$); (2) is an instance of triple; (3) is an instance of naturality of swap.

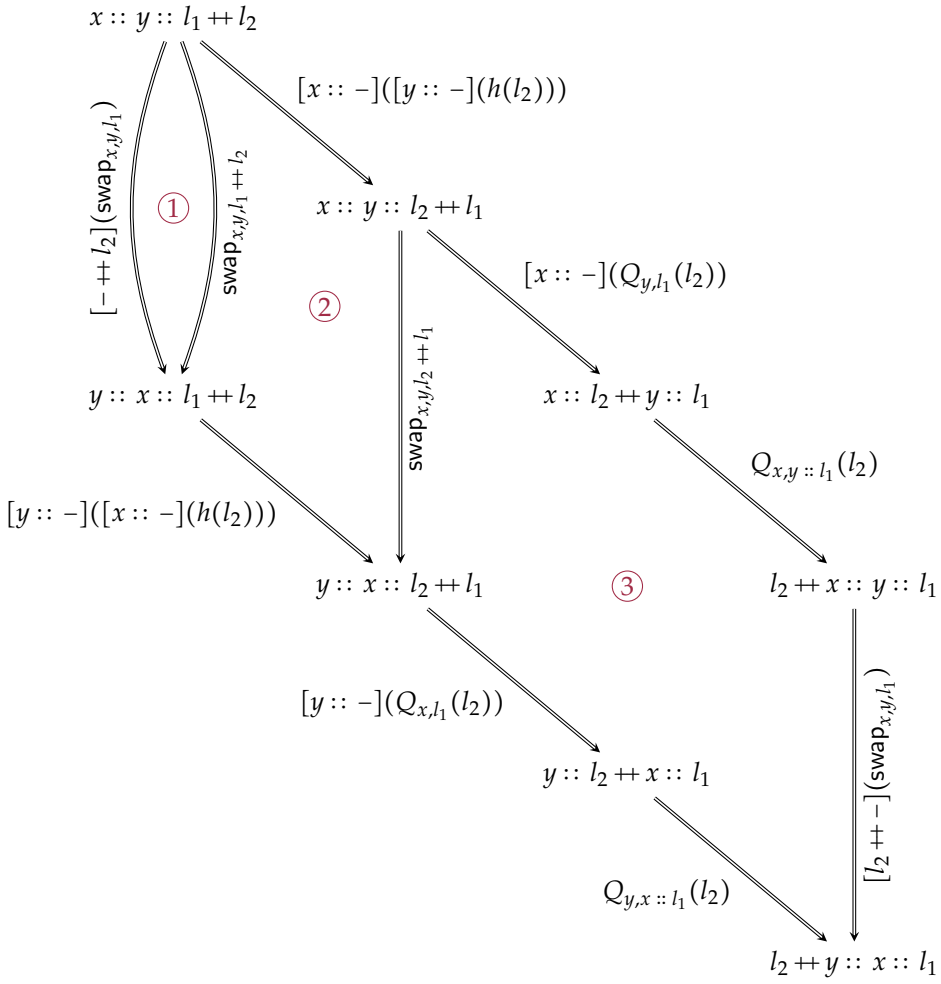


Figure 4.17: The 2-path $\mathsf{swap}'(x, y, l_1, h, l_2)$ in the inductive definition of τ_{slist}, after unfolding the definition of cons'. The 2-path (1) is obtained via (4.20); (2) is an instance of naturality of swap; (3) is given by $R_{x,y,l_1}(l_2)$ defined in Lemma 4.28.

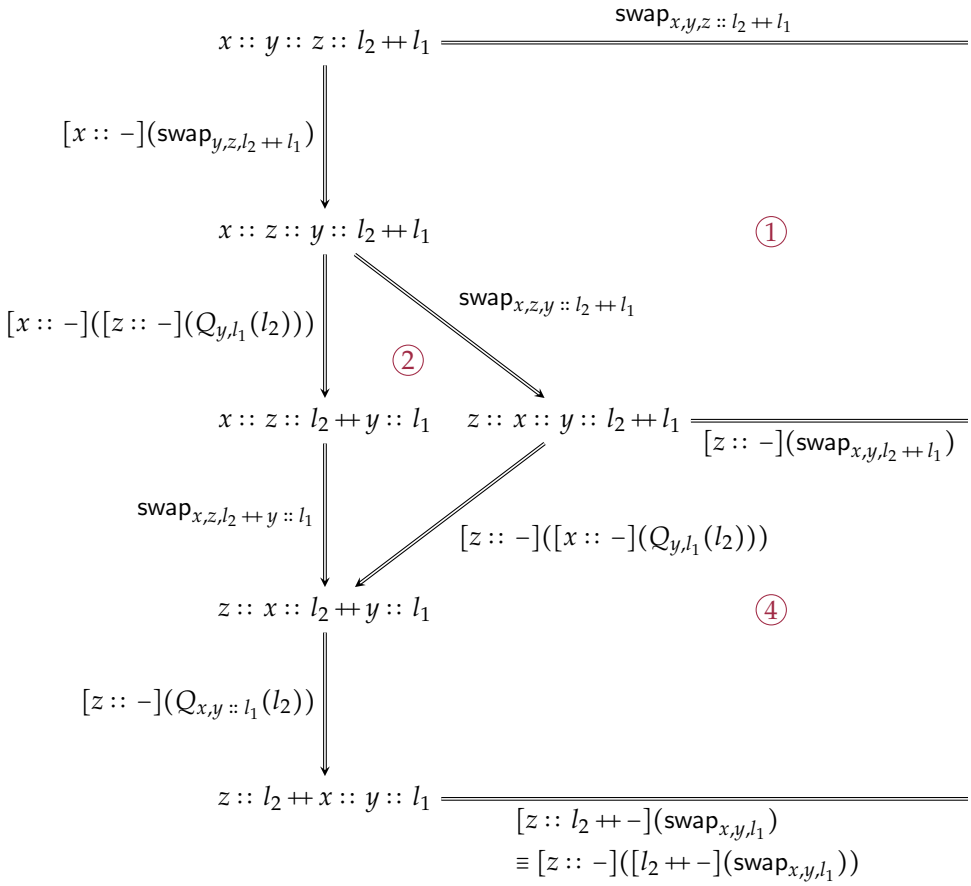
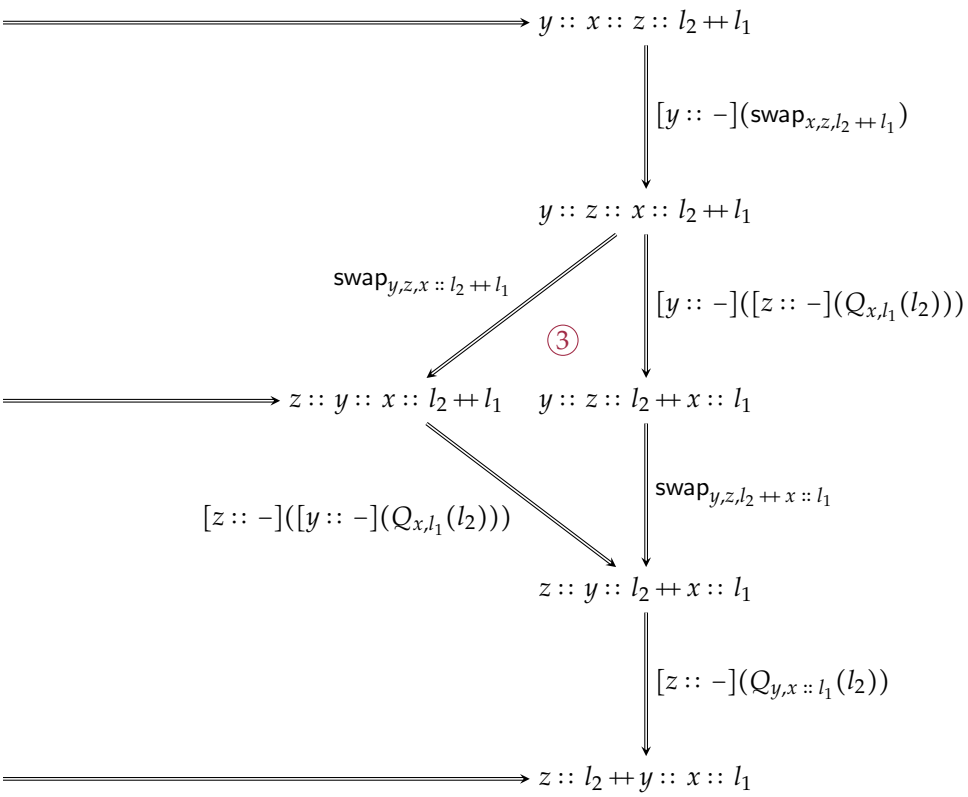
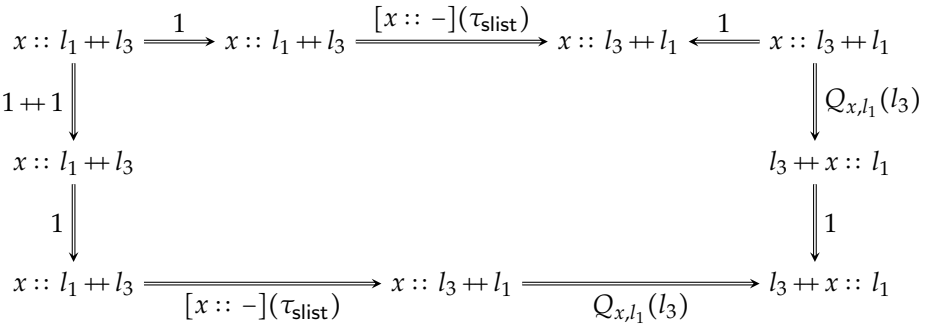
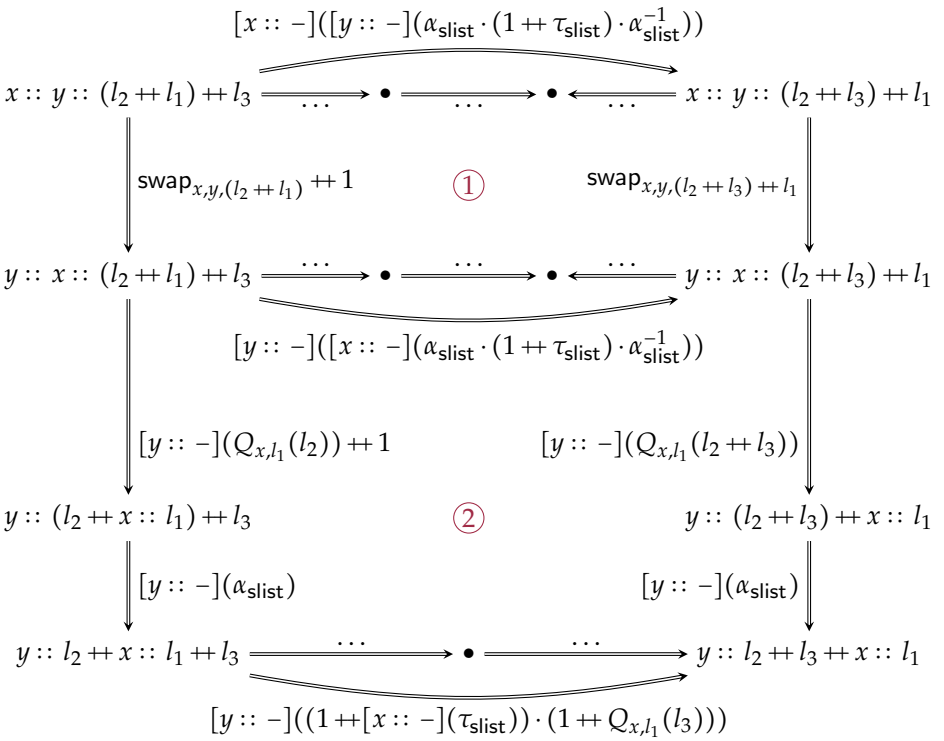


Figure 4.18: The 2-path $\mathsf{cons}'_{x,y,l_1}(z, l_1, h)$ in the inductive definition of R_{x,y,l_1}. The 2-path (1) is an instance of triple; (2) and (3) are instances of naturality of swap; (4) follows from $[[z :: -]](h)$.


$y :: x :: z :: l_2 ++ l_1$
$[y :: -](\mathsf{swap}_{x,z,l_2 ++ l_1})$
$y :: z :: x :: l_2 ++ l_1$
$\mathsf{swap}_{y,z,x :: l_2 ++ l_1}$
$[y :: -]([z :: -](Q_{x,l_1}(l_2)))$
③
$z :: y :: x :: l_2 ++ l_1$
$y :: z :: l_2 ++ x :: l_1$
$\mathsf{swap}_{y,z,l_2 ++ x :: l_1}$
$[z :: -]([y :: -](Q_{x,l_1}(l_2)))$
$z :: y :: l_2 ++ x :: l_1$
$[z :: -](Q_{y,x :: l_1}(l_2))$
$z :: l_2 ++ y :: x :: l_1$

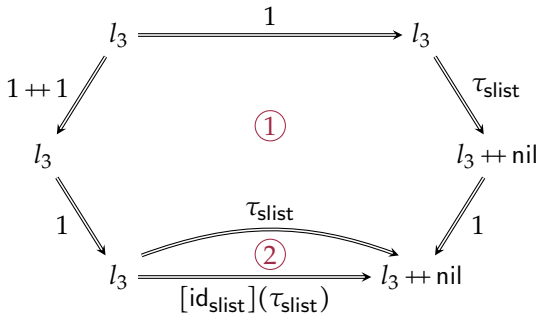


(a) The 2-path $H_{x,l_1,l_3}(\mathsf{nil}) \equiv \mathsf{nil}'_{x,l_1,l_3}$ is trivially obtained, after unfolding the definition of α_{slist} and Q, and using $(\mathsf{refl}_{\mathsf{nil}} ++ p) \equiv [\mathsf{id}_{\mathsf{slist}(X)}](p) = p$ for every suitable p as in (4.16).

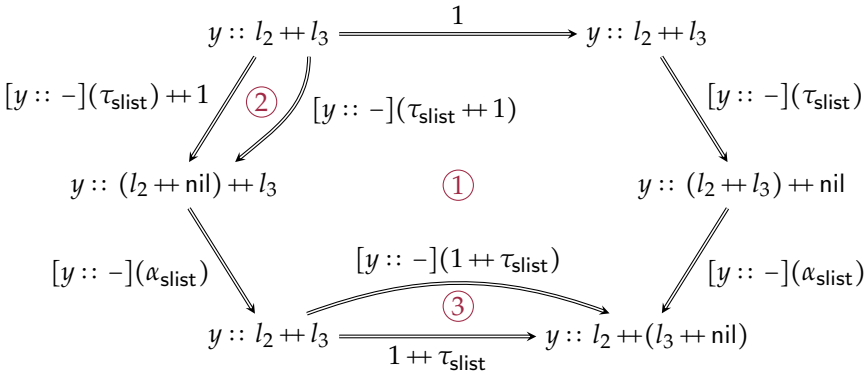


(b) The 2-path $\mathsf{cons}'_{x,l_1,l_3}(y, l_2, h)$, after unfolding the definitions of α_{list} and Q, with some vertices omitted for readability. The 2-path (1) is obtained via naturality of swap, as $\mathsf{swap}_{x,y,(l_2 ++ l_1)} ++ \mathsf{refl}_{l_3} = \mathsf{swap}_{x,y,(l_2 ++ l_1) ++ l_3}$; (2) is derived (recursively) from $[[y :: -]](h)$, using (4.21).

Figure 4.19: Derivation of $H_{x,l_1,l_3}(l_2)$.

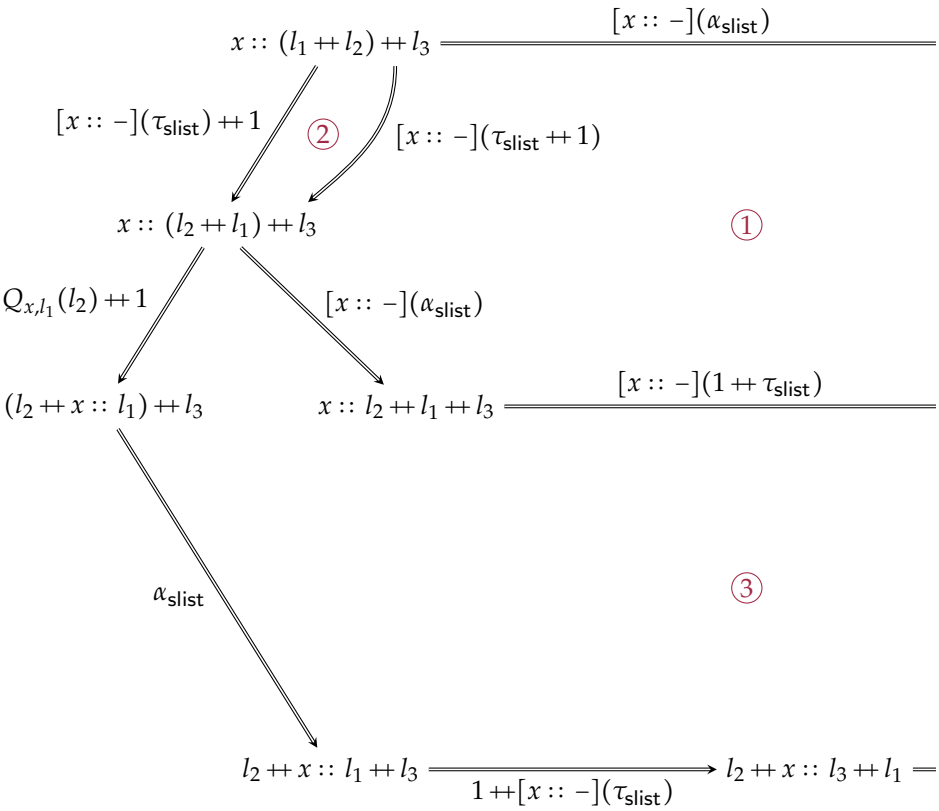


(a) Derivation of the 2-path $\circlearrowright_{\mathsf{slist}}(\mathsf{nil}, \mathsf{nil}, l_3)$, after unfolding the definitions of α_{slist} and τ_{slist}; both instances of the latter are applied to nil and l_3. The 2-path (1) is trivial; (2) is obtained by path algebra.



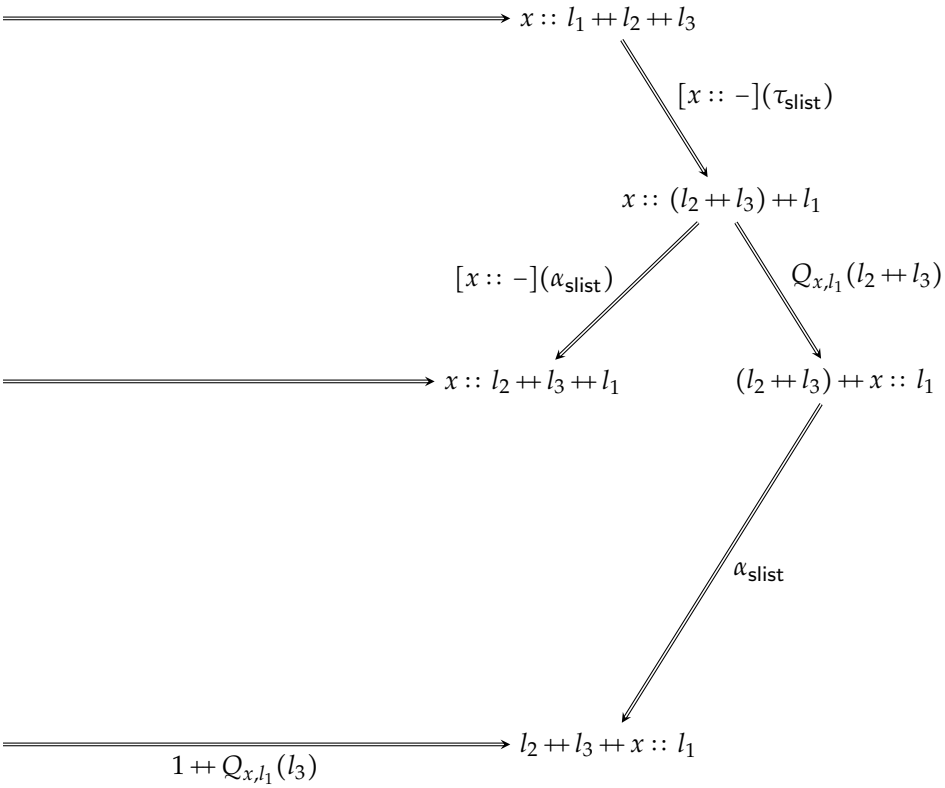
(b) Derivation of the 2-path $\circlearrowright_{\mathsf{slist}}(\mathsf{nil}, y :: l_2, l_3)$, after unfolding the definitions of α_{slist} and τ_{slist}; the latter is applied to nil and $l_2 +\!\!+ l_3$ in the path on the top right, and to nil and l_3 in the path on the bottom of the diagram. The 2-path (1) is given recursively by $[[y :: -]](\circlearrowright_{\mathsf{slist}}(\mathsf{nil}, l_2, l_3))$; (2) and (3) are obtained by path algebra.

Figure 4.20: Derivation of $\circlearrowright_{\mathsf{slist}}$.



(c) Derivation of the 2-path $\bigcirc_{\mathsf{slist}}(x :: l_1, l_2, l_3)$, after unfolding the definitions of α_{slist} and τ_{slist}. The 2-path (1) is given recursively by $[[x :: -]](\bigcirc_{\mathsf{slist}}(l_1, l_2, l_3))$; (2) is obtained by path algebra; (3) is $H_{x,l_1,l_3}(l_2)$ from Lemma 4.32.

Figure 4.20: Continued.


$x :: l_1 ++ l_2 ++ l_3$
$[x :: -](\tau_{\mathsf{slist}})$
$x :: (l_2 ++ l_3) ++ l_1$
$[x :: -](\alpha_{\mathsf{slist}})$
$Q_{x,l_1}(l_2 ++ l_3)$
$x :: l_2 ++ l_3 ++ l_1$
$(l_2 ++ l_3) ++ x :: l_1$
α_{slist}
$l_2 ++ l_3 ++ x :: l_1$
$1 ++ Q_{x,l_1}(l_3)$

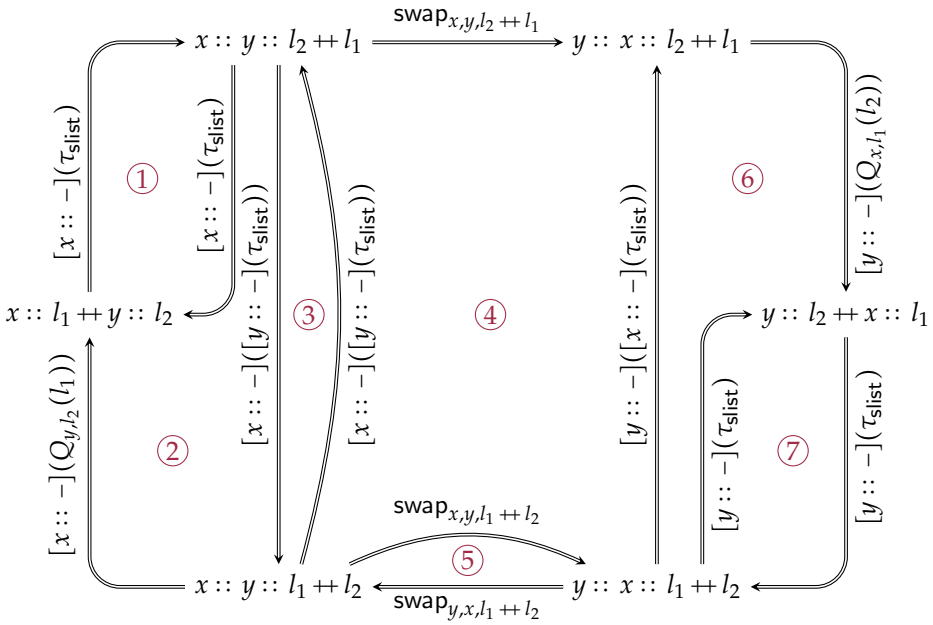


Figure 4.21: Derivation of the 2-path $\diamond_{\mathsf{slist}}(x :: l_1, y :: l_2)$, after unfolding the definition of τ_{slist}. The 2-path (1) is obtained recursively by $[[x :: -]](\diamond_{\mathsf{slist}}(l_1, y :: l_2))$; (2) and (6) are derived from the definition of τ_{slist}; (3) is given recursively using $\diamond_{\mathsf{slist}}(l_1, l_2)$; (4) is an instance of naturality of swap; (5) is an instance of double; (7) is obtained recursively from $[[y :: -]](\diamond_{\mathsf{slist}}(x :: l_1, l_2))$.

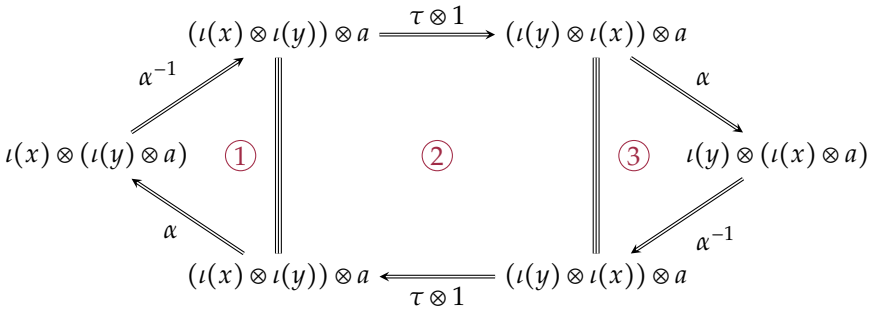


Figure 4.22: The 2-path $\mathsf{double}'_{x,y,a}$ in the inductive definition of J, after unfolding the definition of swap'. The 2-paths (1) and (3) are trivial; (2) is derived from $\diamond \otimes \mathsf{refl}_a$.

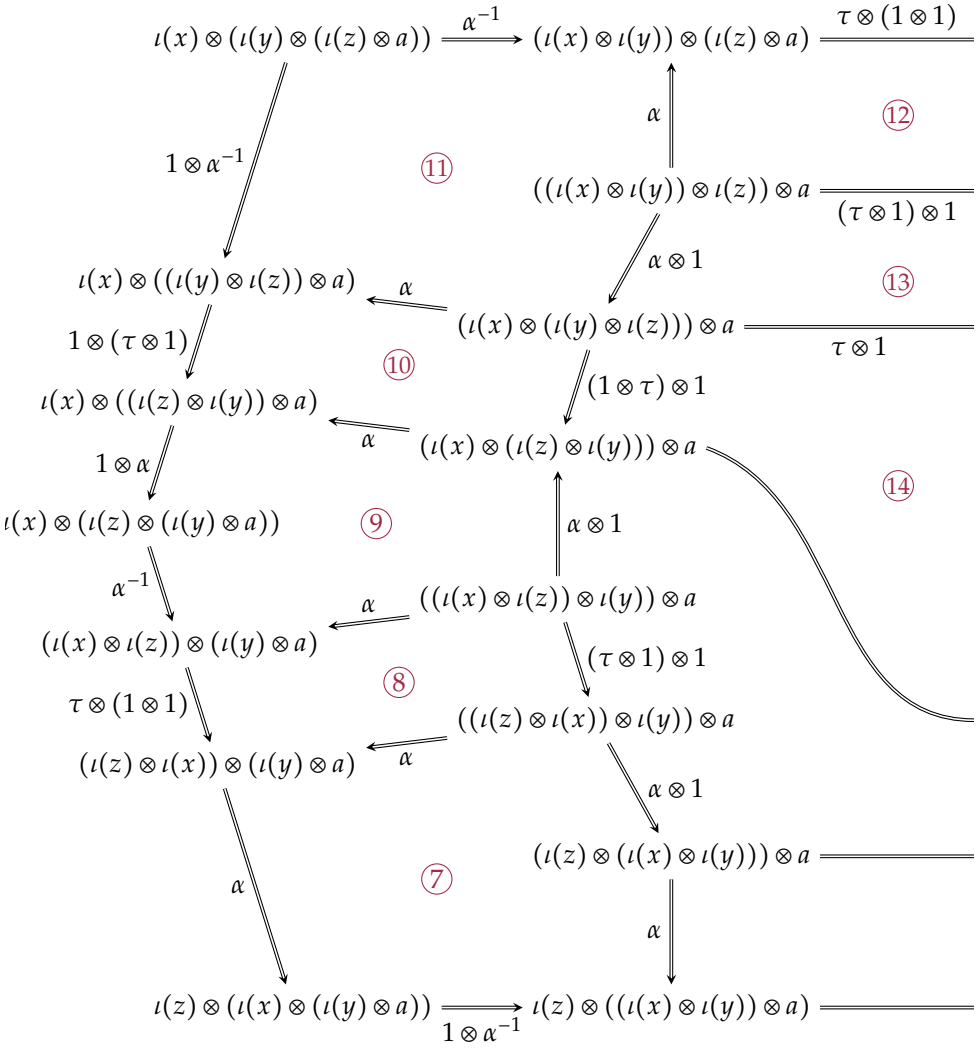
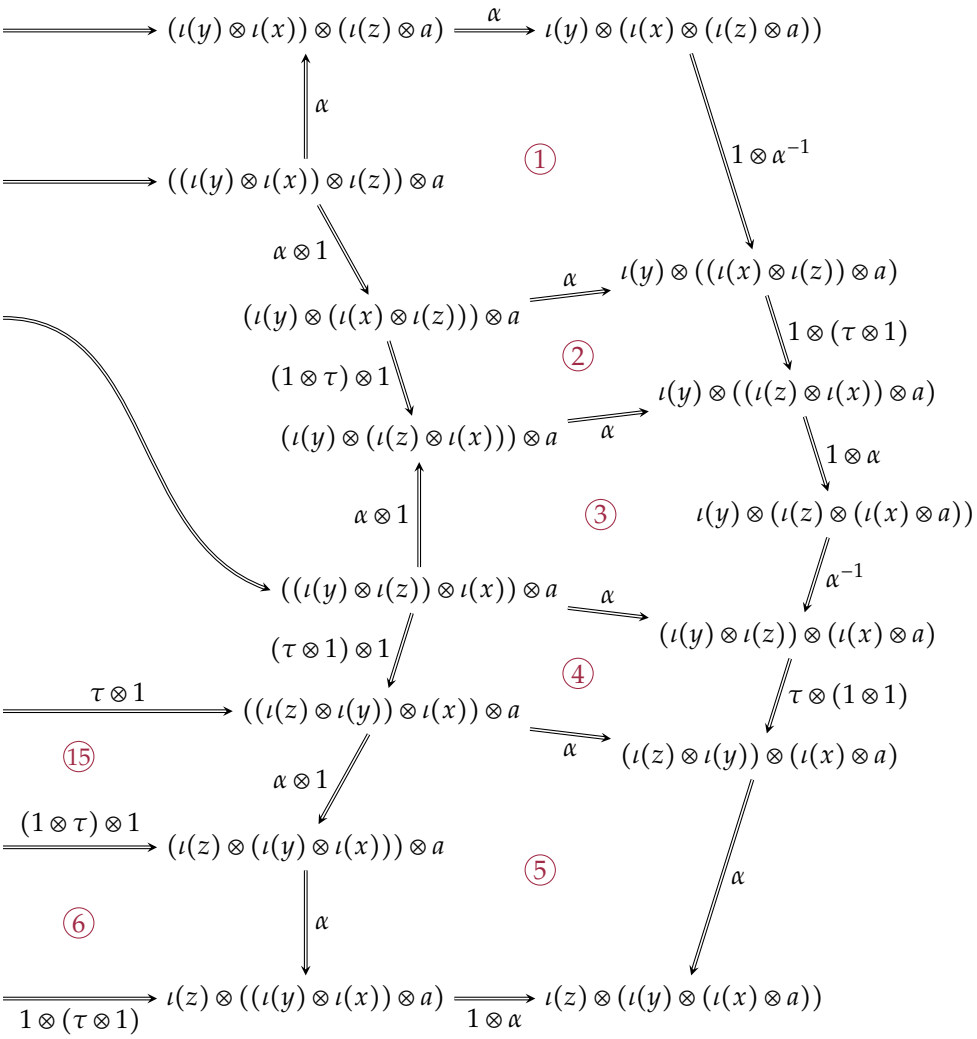


Figure 4.23: The 2-path $\mathsf{triple}'_{x,y,z,a}$ in the inductive definition of J, after unfolding the definitions of swap' and cons'. Outer ring: the odd-numbered 2-paths are instances of $\pentagon$; the even-numbered 2-paths are instances of naturality of α. Center: (13) and (15) are derived from $\pentagon \otimes \mathsf{refl}_a$; (14) is obtained by naturality of τ.


$(\iota(y)\otimes\iota(x))\otimes(\iota(z)\otimes a)$
α
$\iota(y)\otimes(\iota(x)\otimes(\iota(z)\otimes a))$
α
$1\otimes\alpha^{-1}$
$((\iota(y)\otimes\iota(x))\otimes\iota(z))\otimes a$
①
$\alpha\otimes 1$
$\iota(y)\otimes((\iota(x)\otimes\iota(z))\otimes a)$
α
$(\iota(y)\otimes(\iota(x)\otimes\iota(z)))\otimes a$
$1\otimes(\tau\otimes 1)$
②
$(1\otimes\tau)\otimes 1$
$\iota(y)\otimes((\iota(z)\otimes\iota(x))\otimes a)$
α
$(\iota(y)\otimes(\iota(z)\otimes\iota(x)))\otimes a$
$1\otimes\alpha$
$\alpha\otimes 1$
③
$\iota(y)\otimes(\iota(z)\otimes(\iota(x)\otimes a))$
α^{-1}
$((\iota(y)\otimes\iota(z))\otimes\iota(x))\otimes a$
α
$(\iota(y)\otimes\iota(z))\otimes(\iota(x)\otimes a)$
$(\tau\otimes 1)\otimes 1$
④
$\tau\otimes(1\otimes 1)$
$\tau\otimes 1$
$((\iota(z)\otimes\iota(y))\otimes\iota(x))\otimes a$
α
$(\iota(z)\otimes\iota(y))\otimes(\iota(x)\otimes a)$
⑮
$\alpha\otimes 1$
$(1\otimes\tau)\otimes 1$
$(\iota(z)\otimes(\iota(y)\otimes\iota(x)))\otimes a$
⑤
α
⑥
α
$\iota(z)\otimes((\iota(y)\otimes\iota(x))\otimes a)$
$\iota(z)\otimes(\iota(y)\otimes(\iota(x)\otimes a))$
$1\otimes(\tau\otimes 1)$
$1\otimes\alpha$

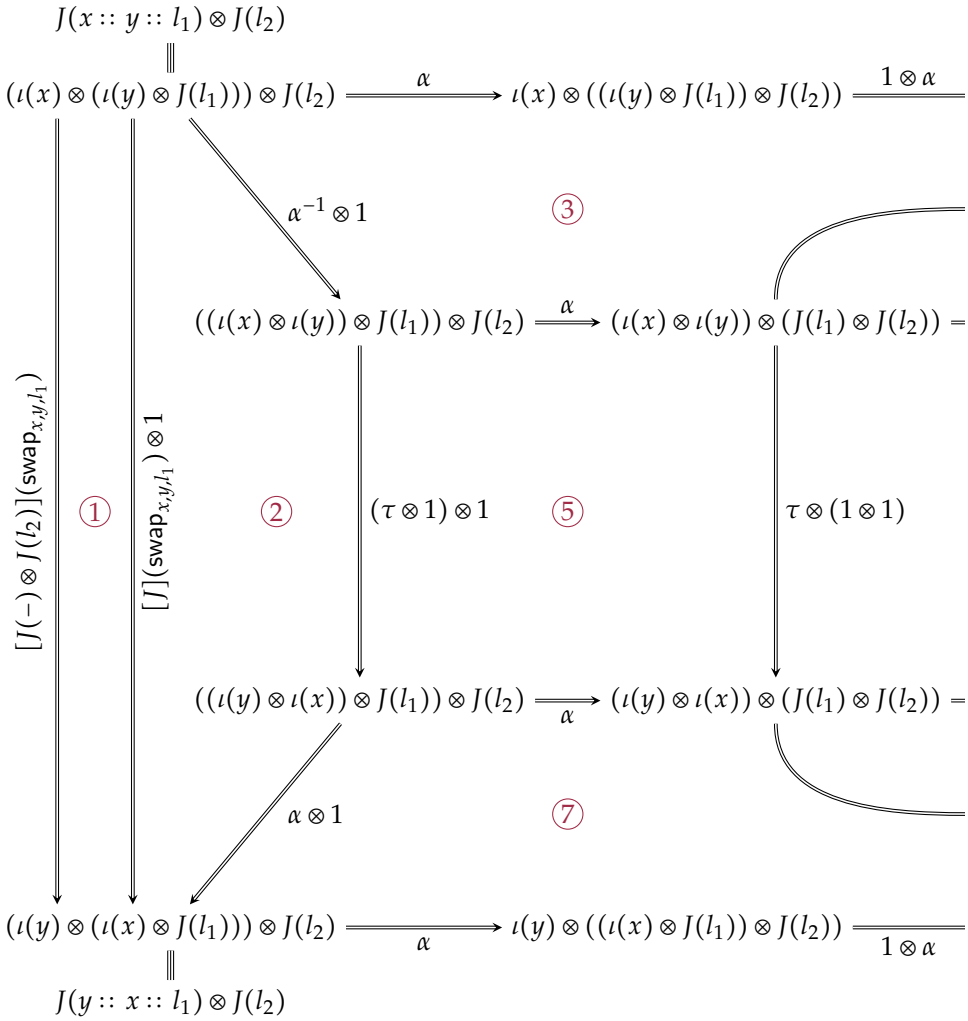
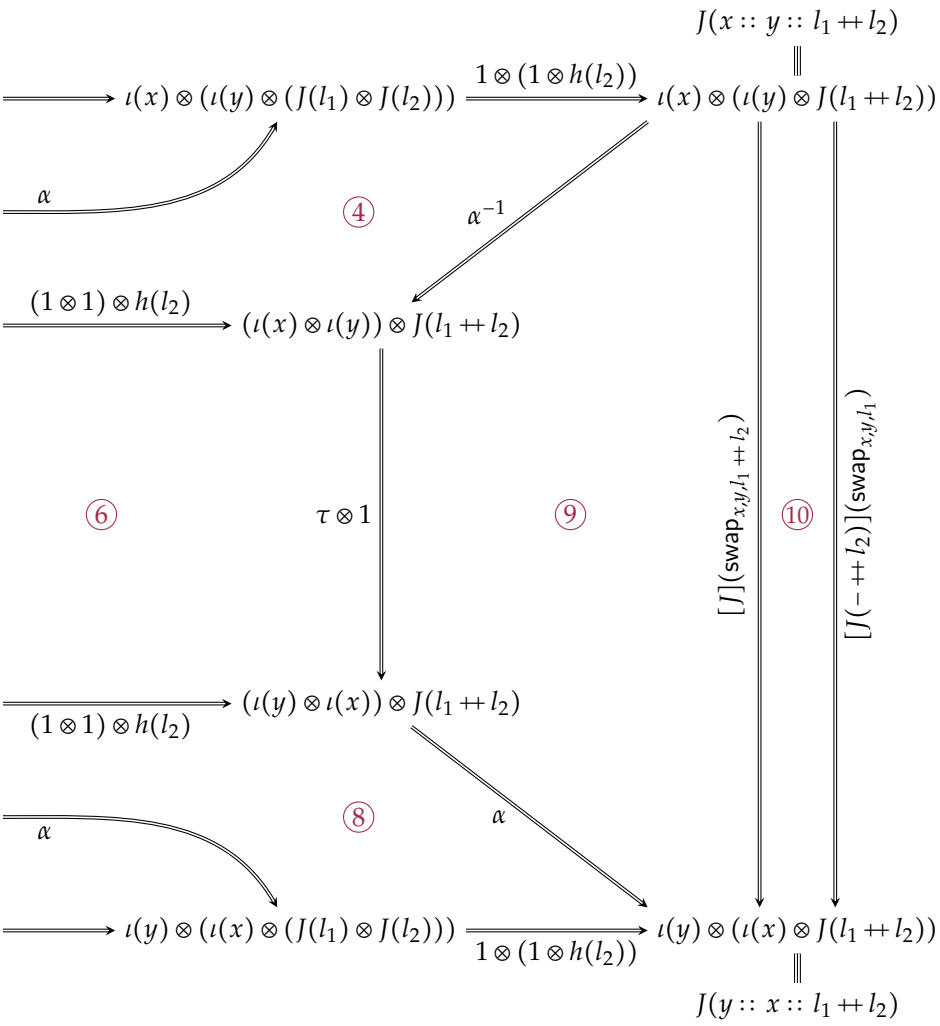
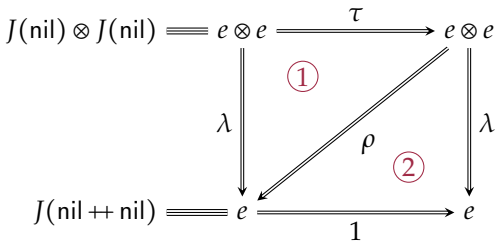
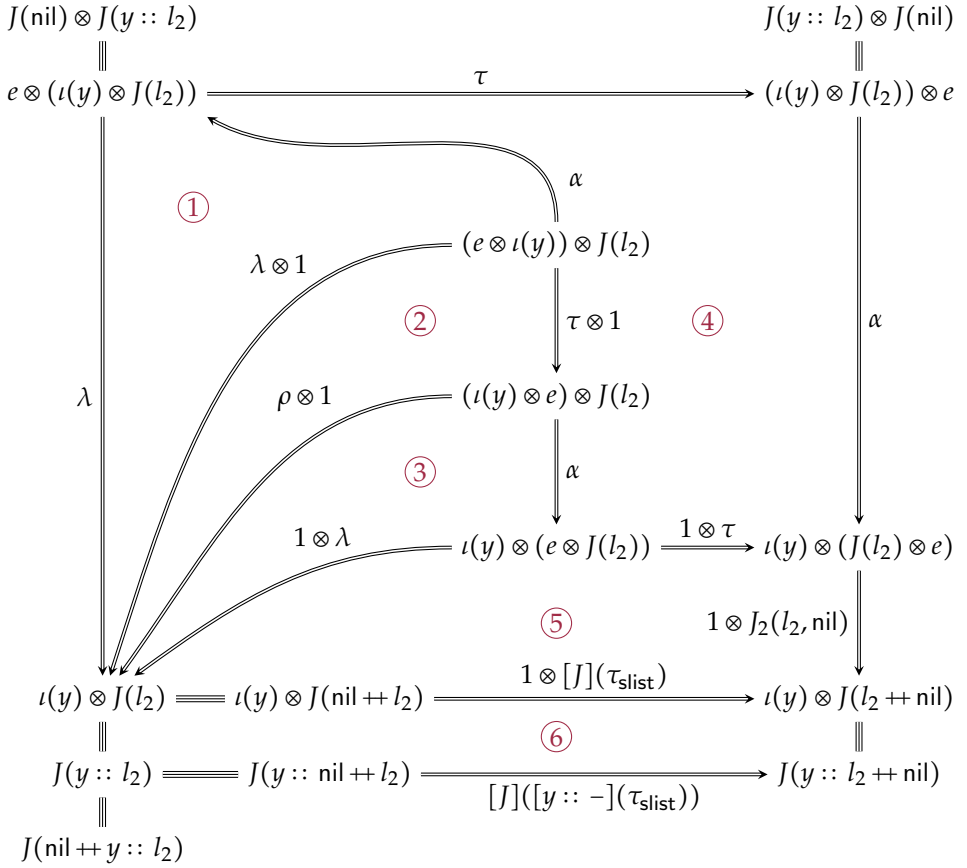


Figure 4.24: The 2-path $\mathsf{swap}'_{x,y,l_1,h,l_2}$ in the inductive definition of J_2, after unfolding the definition of cons'. The 2-paths (1) is obtained by path algebra; (2) and (9) are filled by the computation rule (4.44) of J; (3) and (7) are instances of $\pentagon$; (4), (5) and (8) are instances of naturality of α; (6) is given by the interchange law (2.64); (10) is given by path algebra and (4.20).


$J(x :: y :: l_1 +\!\!+ l_2)$
$\iota(x) \otimes (\iota(y) \otimes (J(l_1) \otimes J(l_2)))$
$1 \otimes (1 \otimes h(l_2))$
$\iota(x) \otimes (\iota(y) \otimes J(l_1 +\!\!+ l_2))$
α
④
α^{-1}
$(1 \otimes 1) \otimes h(l_2)$
$(\iota(x) \otimes \iota(y)) \otimes J(l_1 +\!\!+ l_2)$
⑥
$\tau \otimes 1$
⑨
⑩
$[J](\mathsf{swap}_{x,y,l_1 +\!\!+ l_2})$
$[J(- +\!\!+ l_2)](\mathsf{swap}_{x,y,l_1})$
$(\iota(y) \otimes \iota(x)) \otimes J(l_1 +\!\!+ l_2)$
$(1 \otimes 1) \otimes h(l_2)$
⑧
α
α
$\iota(y) \otimes (\iota(x) \otimes (J(l_1) \otimes J(l_2)))$
$1 \otimes (1 \otimes h(l_2))$
$\iota(y) \otimes (\iota(x) \otimes J(l_1 +\!\!+ l_2))$
$J(y :: x :: l_1 +\!\!+ l_2)$

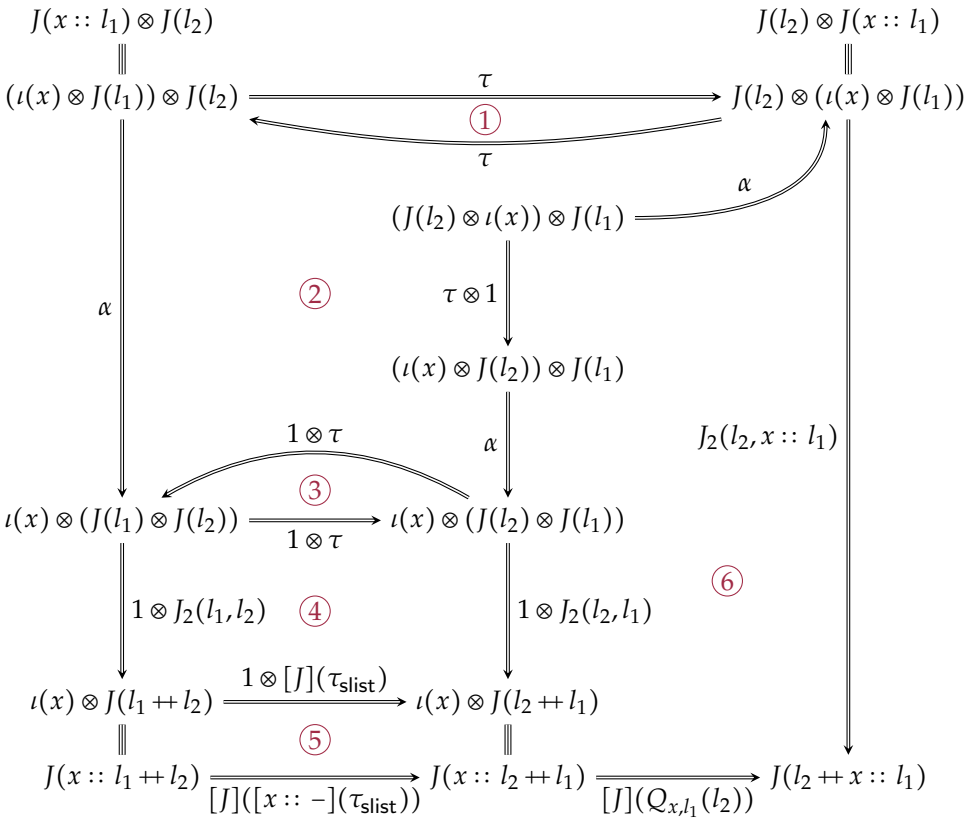


(a) Derivation of the 2-path $J_\tau(\mathsf{nil}, \mathsf{nil})$. The 2-path (1) is an instance of the diagram in Fig. 4.4b (Lemma 4.6); (2) is the diagram in Fig. 3.4c (Remark 4.4).



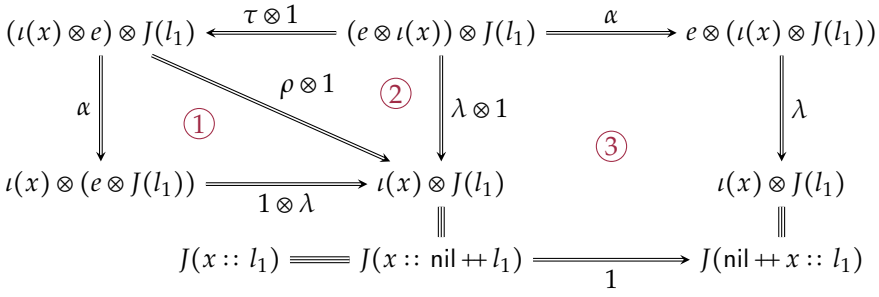
(b) Derivation of the 2-path $J_\tau(\mathsf{nil}, y :: l_2)$, after unfolding the definitions of J, J_2 and τ_{slist}. The 2-path (1) is an instance of the diagram in Fig. 3.4a (Remark 4.4); (2) is obtained via the diagram in Fig. 4.4b (Lemma 4.6); (3) is an instance of $\triangledown$; (4) is an instance of $\bigcirc$; (5) is obtained, recursively, by $1 \otimes J_\tau(\mathsf{nil}, l_2)$; (6) is obtained by path algebra.

Figure 4.25: Derivation of the 2-path $J_\tau(l_1, l_2)$, by induction on l_1 and l_2.



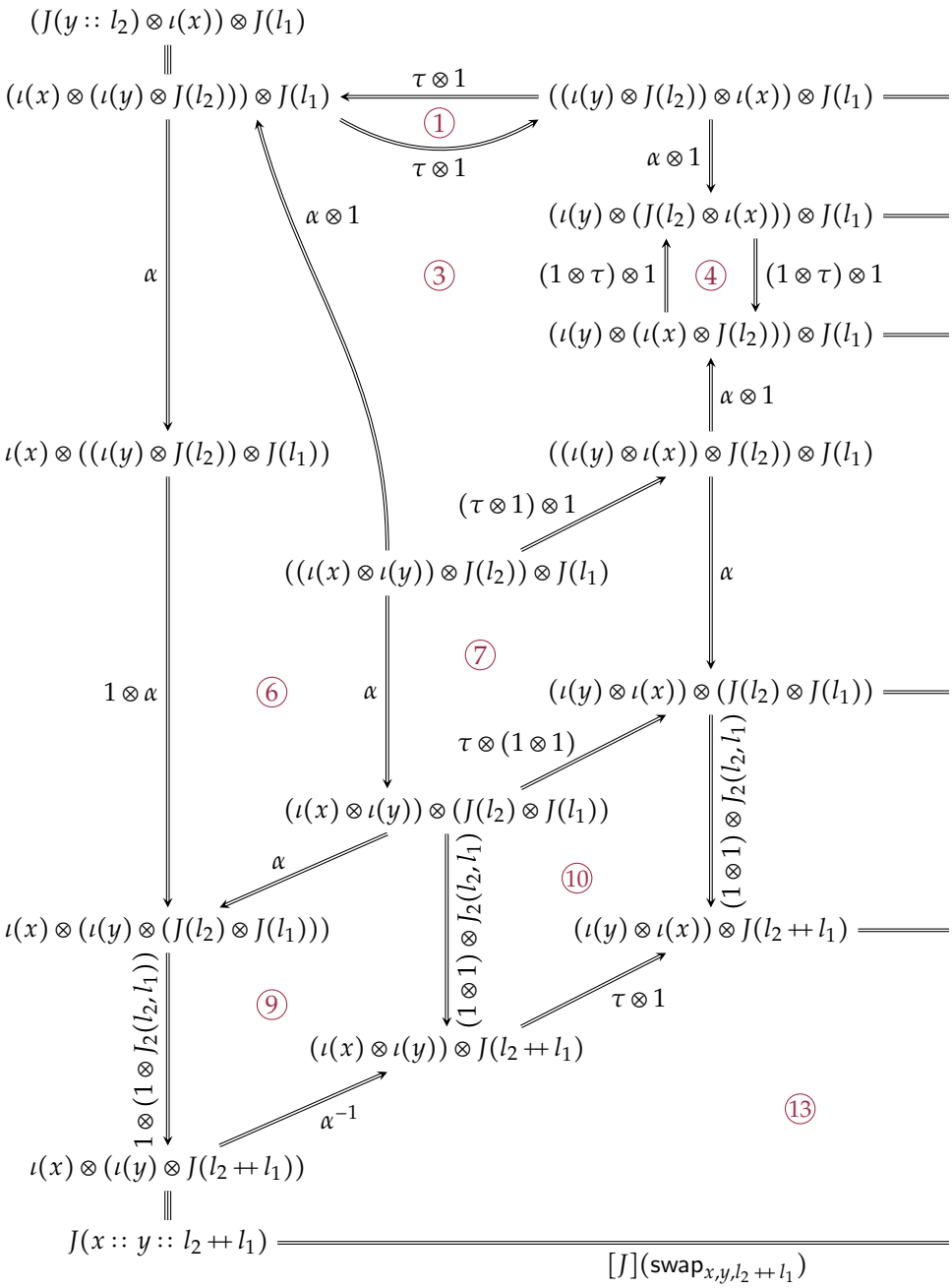
(c) Derivation of the 2-path $J_\tau(x :: l_1, l_2)$, after unfolding the definitions of J, J_2 and τ_{slist}. The 2-paths (1) and (3) are derived from $\hookrightarrow$; (2) is an instance of $\circlearrowright$; (4) is obtained, recursively, from $1 \otimes J_\tau(l_1, l_2)$; (5) is path algebra; (6) is given by $V_{x,l_1}(l_2)$, filled by induction on l_2 (Fig. 4.26).

Figure 4.25: Continued.



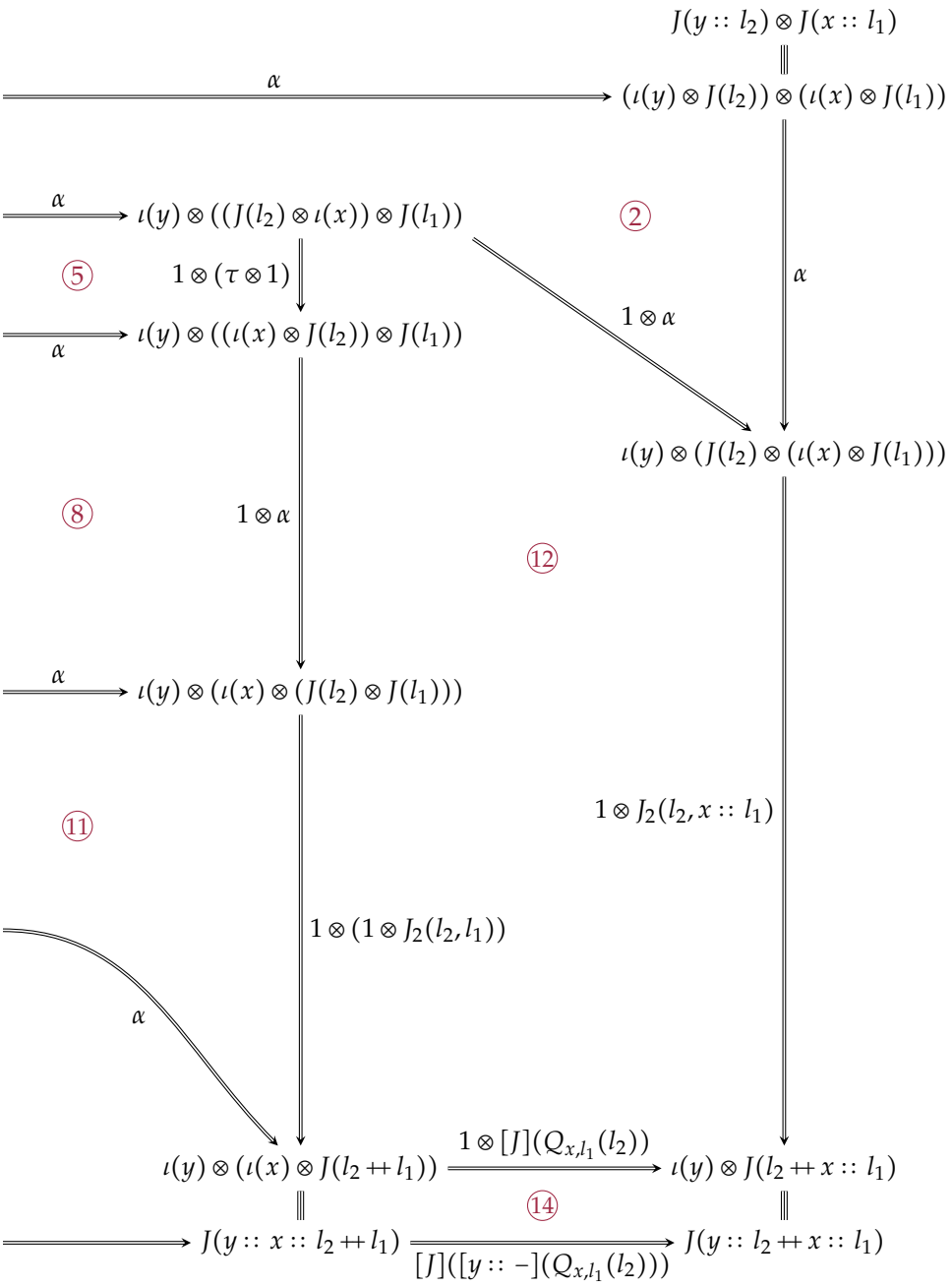
(a) Derivation of the 2-path $V_{x,l_1}(\mathsf{nil})$, after unfolding the definitions of J, J_2 and Q. The 2-path (1) is an instance of $\triangledown$; (2) is derived from the diagram in Fig. 4.4b (Lemma 4.6); (3) is an instance of the diagram in Fig. 3.4a (Remark 4.4).

Figure 4.26: Derivation of the 2-path $V_{x,l_1}(l_2)$ by induction on l_2.



(b) Derivation of the 2-path $V_{x,l_1}(y :: l_2)$, after unfolding the definitions of J, J_2 and Q. The 2-paths (1) and (4) are obtained from $\diamond$; (2), (6) and (8) from $\pentagon$; (3) from $\bigcirc$; (5), (7), (9) and (11) from naturality of α; (10) from (2.64); (12) from $1 \otimes V_{x,l_1}(l_2)$; (13) by computation rule of J; (14) by path algebra.

Figure 4.26: Continued.


$J(y :: l_2) \otimes J(x :: l_1)$
α
$(\iota(y) \otimes J(l_2)) \otimes (\iota(x) \otimes J(l_1))$
α
$\iota(y) \otimes ((J(l_2) \otimes \iota(x)) \otimes J(l_1))$
2
5
$1 \otimes (\tau \otimes 1)$
α
$1 \otimes \alpha$
α
$\iota(y) \otimes ((\iota(x) \otimes J(l_2)) \otimes J(l_1))$
$\iota(y) \otimes (J(l_2) \otimes (\iota(x) \otimes J(l_1)))$
8
$1 \otimes \alpha$
12
α
$\iota(y) \otimes (\iota(x) \otimes (J(l_2) \otimes J(l_1)))$
$1 \otimes J_2(l_2, x :: l_1)$
11
$1 \otimes (1 \otimes J_2(l_2, l_1))$
α
$1 \otimes [J](Q_{x,l_1}(l_2))$
$\iota(y) \otimes (\iota(x) \otimes J(l_2 +\!\!+ l_1))$
$\iota(y) \otimes J(l_2 +\!\!+ x :: l_1)$
14
$J(y :: x :: l_2 +\!\!+ l_1)$
$[J]([y :: -](Q_{x,l_1}(l_2)))$
$J(y :: l_2 +\!\!+ x :: l_1)$

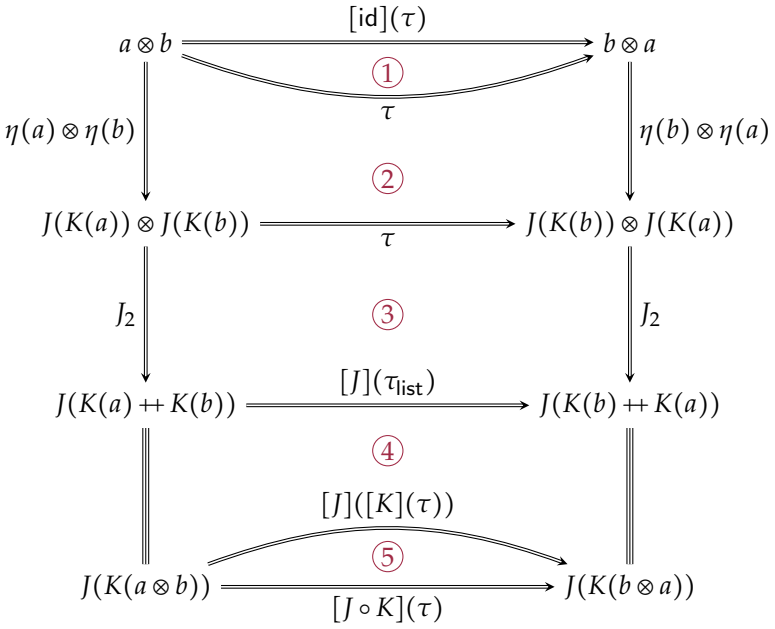


Figure 4.27: The diagram corresponding to τ' in the inductive definition of η. The 2-paths (1) and (5) are given by path algebra; (2) is an instance of naturality of τ; (3) is an instance of J_τ; (4) is given by a computation rule of K.

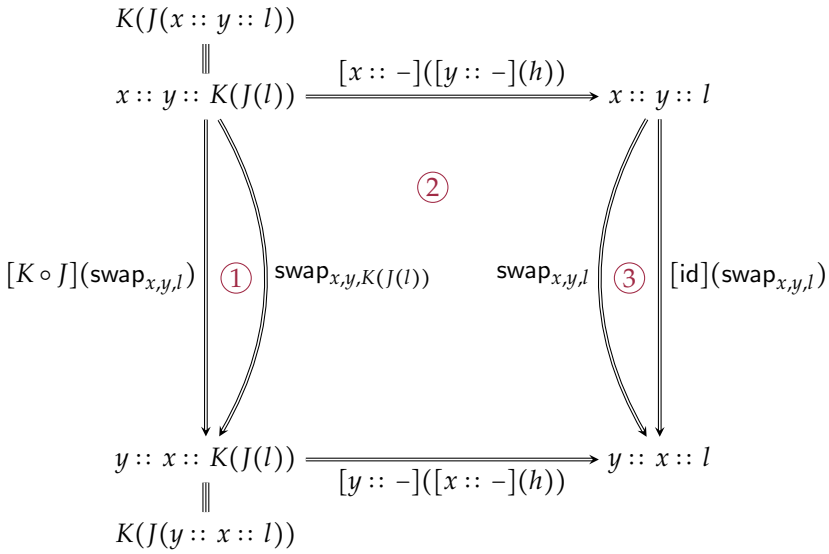


Figure 4.28: The 2-path $\mathsf{swap}'_{x,y,l,h}$ in the inductive definition of ϵ. The 2-path (1) is given in (4.48); (2) is an instance of naturality of swap; (3) is obtained by path algebra.

Chapter 5

Finite Types and Symmetric Monoidal Structures

The definition of the HIT construction slist in Chapter 4 was motivated by the relationship between symmetric monoidal structures and symmetric groups. Informally, given a list of n elements chosen from a set X, we can let any element of the symmetric group S_n act on it and obtain another list, which is then identified with the original one.

Groups can be defined as HITs in HoTT. For instance, the free group over a 0-type X can be defined similarly to the type $\mathsf{FM}(X)$ in Section 3.1, by adding a 0-constructor for the inverse operation, 1-constructors for the inverse laws, and a 0-truncation [see Uni13, Chapter 6]. In general, the presentation of a group suggests which 0- and 1-constructors are to be included in its definition as a HIT: the former should correspond to the generators, while the latter to the relations that the generators should satisfy. Indeed, the symmetric groups S_n could be constructed as a family of (0-truncated) 1-HITs in this way, according to their presentation given in (4.10); one could specify their action on lists and *then* construct the quotient of $\mathsf{list}(X)$ by the group action, also as a 1-HIT, with 1-constructors identifying lists in the same orbit. This process is unnecessarily convoluted; the constructors we gave for $\mathsf{slist}(X)$ bypass this intermediate step and make all the required identifications hardwired in its definition. The only price to pay is that $\mathsf{slist}(X)$ is a 2-HIT, where the 2-constructors account for the identifications of the elements in the group.

In principle, the shortcut we just described holds for the quotient of any set by the action of any group. Symmetric groups, however, have an alternative description: their elements correspond to automorphisms $\sigma : [n] \simeq [n]$ of canonical finite types. We saw in Section 3.7 how a list with elements in a 0-type X can be described as a vector $v : [n] \to X$ for some $n : \mathbb{N}$, i.e., as an ordered choice of finitely many (and possibly repeating) terms in X. The symmetric group acts on vectors by precomposition; accordingly, a *symmetric* list of elements in X ought to be determined

by a choice of finitely many terms in X in a strictly *non-specified* order, i.e., as a term in the quotient of the function type $[n] \to X$ by a relation identifying any $v : [n] \to X$ with $v \circ \sigma$, for any $\sigma : [n] \simeq [n]$, for some $n : \mathbb{N}$.

A way to circumvent the employment of quotients is to describe symmetric lists as functions $v : A \to X$, where A is a type belonging to the *classifying space* $\mathcal{B}\mathsf{S}_n$ of the symmetric group S_n, for some $n : \mathbb{N}$. Such a type A is a finite type whose terms are not labelled in any order, and hence a function $v : A \to X$ incorporates in its source the features of the sought quotient: instead of identifying all the ways of arranging the elements in a list, it removes the ordering altogether from the picture.

Following the discussion above, and parallel to the chain of monoidal equivalences

$$\mathsf{FMG}(X) \simeq \mathsf{list}(X) \simeq \Sigma\,(n : \mathbb{N})\,.\,([n] \to X)$$

in (3.59), what we would like to show is a chain of symmetric monoidal equivalences between the type of free symmetric monoidal expressions with elements in a 0-type X and the type of unordered, finite vectors with entries in X, via the type of symmetric lists in X; that is,

$$\mathsf{FSMG}(X) \simeq \mathsf{slist}(X) \simeq \Sigma\,(A : \mathcal{B}\mathsf{S}_\bullet)\,.\,(A \to X) \tag{5.1}$$

where $\mathcal{B}\mathsf{S}_\bullet$ is the subuniverse of finite types, which we will describe in Section 5.1. The chain of equivalences in (5.1) (of which the leftmost equivalence has already been achieved in Corollary 4.49, while the rightmost will be presented in Corollary 5.103) would give us an elementary way to characterize a class of commuting diagrams in $\mathsf{FSMG}(X)$ – namely, the diagrams involving symmetric monoidal expressions with no repetitions. While these expressions are hard to pin down in $\mathsf{FSMG}(X)$, they simply correspond to vectors in $\Sigma\,(A : \mathcal{B}\mathsf{S}_\bullet)\,.\,(A \to X)$ which are *embeddings* (Theorem 5.104).

We will start by turning our attention to the simple case where $X :\equiv \mathbf{1}$ in (5.1), and work our way through a symmetric monoidal equivalence

$$\mathsf{slist}(\mathbf{1}) \simeq \mathcal{B}\mathsf{S}_\bullet. \tag{5.2}$$

The formalization of the equivalence in (5.2), discussed in this chapter, is not complete. This is due to the fact that, in order to construct an equivalence $\mathsf{slist}(\mathbf{1}) \simeq \mathcal{B}\mathsf{S}_\bullet$, we need to delve into the combinatorics of finite types, essentially equating the two mentioned descriptions of the symmetric groups (as automorphisms of finite types, or in their presentation with generators and relations); such a task revealed itself

to be, in our framework, rather complex. However, the equivalence in (5.2) holds under a very limited amount of unformalized results, which we collect in Assumption 5.91; the nature of these deficiencies will be discussed there. In addition, we reach a number of interesting results; particularly, we define an indexed HIT $\mathsf{del}_\bullet$ of *deloopings* of symmetric groups, which acts as a bridge between symmetric lists and finite types in the equivalence.

Univalence is assumed throughout this chapter. Selected parts of the formalization are featured in Appendix A.3.

5.1 Finite Types

This section discusses the subuniverse of finite types, which is an already established notion in HoTT. We will start from its definition.

Definition 5.3 (Finite types)**.** A type $A : \mathcal{U}$ together with a term $t : \|A \simeq [n]\|$ for some $n : \mathbb{N}$ is called a **finite type** and the term n its **cardinality**. We define the type family $\mathcal{B}\mathsf{S} : \mathbb{N} \to \mathcal{U}$ by declaring

$$\mathcal{B}\mathsf{S}_n :\equiv \Sigma\,(A : \mathcal{U})\,.\,\|A \simeq [n]\|$$

for every $n : \mathbb{N}$, i.e., $\mathcal{B}\mathsf{S}_n$ is the subuniverse of finite types of cardinality n. The subuniverse[1] of finite types of any cardinality will be denoted by

$$\mathcal{B}\mathsf{S}_\bullet :\equiv \Sigma\,(n : \mathbb{N})\,.\,\mathcal{B}\mathsf{S}_n.$$

The chosen notation "$\mathcal{B}\mathsf{S}_n$" borrows the character $\mathcal{B}$ from the usual notation for the classifying space of a group, which in this case is the group $\mathsf{S}_n :\equiv \mathrm{Aut}([n])$ of automorphisms of the (canonical) finite type $[n]$; we use this notation to emphasize a view of the subuniverse of finite types of cardinality n as a *classifying type* of $\mathrm{Aut}([n])$, which is, indeed, the subuniverse of those types which are anonymously equivalent to $[n]$. A discussion and examples on classifying types in HoTT can be found in [Shu15] and [BvDR18].

As hinted in the introduction to this chapter, an important difference runs between finite types $\mathcal{B}\mathsf{S}_n$ and *canonical* finite types $[n]$. Induction on the terms of a coproduct allows to *count* the terms in a canonical finite type: for example, one can

[1] To keep true to Definition 2.73, the subuniverse $\mathcal{B}\mathsf{S}_\bullet$ of finite types should be given by $\Sigma\,(A : \mathcal{U})\,.\,\Sigma\,(n : \mathbb{N})\,.\,\|A \simeq [n]\|$, where the inner Σ-type is provably a (-1)-type. The two definitions are equivalent.

prove that no more than three "distinct" terms inhabit the type $[3] \equiv \mathbf{0} + \mathbf{1} + \mathbf{1} + \mathbf{1}$, i.e., given terms $a_1, a_2, a_3, a_4 : [3]$ and functions $(a_1 = a_2) \to \mathbf{0}$, $(a_1 = a_3) \to \mathbf{0}$, ..., a term in $\mathbf{0}$ can be obtained. The inductive definition of $[n+1]$ as $[n] + \mathbf{1}$ determines a "system of coordinates" on its terms: in the example above, the (only) terms inhabiting $[3]$ are $\mathsf{inr}(*)$, $\mathsf{inl}(\mathsf{inr}(*))$ and $\mathsf{inl}(\mathsf{inl}(\mathsf{inr}(*)))$.

Finite types diverge from canonical finite types in that we do not have access to their elements: given a term $X : \mathcal{BS}_3$, it is still possible to prove that $\mathsf{pr}_1(X) : \mathcal{U}$ is inhabited by no more than three different terms, but no coordinate system is given – by design, it cannot be inherited by the term $\mathsf{pr}_2(X) : \|\mathsf{pr}_1(X) \simeq [3]\|$. However, all *properties* (in the sense of Definition 2.73) of canonical finite types can be transported to finite types, in the following sense.

Lemma 5.4. *Given a family $P : \mathcal{U} \to \mathcal{U}$ of types, a term in $\prod(X:\mathcal{U}).\, \mathsf{IsHProp}(P(X))$ and a term $n : \mathbb{N}$, there is a function*

$$\prod(X : \mathcal{BS}_n).\, (P(\mathsf{pr}_1(X)) \simeq P([n])).$$

Proof. By the elimination principle of Σ-types, it is enough to provide, for every $A : \mathcal{U}$, a function $\|A \simeq [n]\| \to (P(A) \simeq P([n]))$. As the target type is a (-1)-type, it is enough to exhibit a function $(A \simeq [n]) \to (P(A) \simeq P([n]))$, which is given by $(e \mapsto (\mathsf{ua}(e))^P_*)$. □

Corollary 5.5. *Let $n : \mathbb{N}$ and $X : \mathcal{BS}_n$. The type $\mathsf{pr}_1(X) : \mathcal{U}$ is a 0-type; i.e., any finite type is a 0-type.*

Proof. Any canonical finite type $[n]$ is a 0-type, as one can prove by induction on n, since $\mathbf{0}$, $\mathbf{1}$ and coproducts of 0-types are 0-types (Remark 2.75). The result then follows from Lemma 5.4, because $\mathsf{IsHSet} : \mathcal{U} \to \mathcal{U}$ is a property (Lemma 2.77). □

More considerations can be made about the truncation level of the subuniverse of finite types itself.

Lemma 5.6. *Let $n : \mathbb{N}$, and let A and B be finite types of cardinality n. The identity type $(A =_{\mathcal{U}} B)$ is a 0-type.*

Proof. By univalence, the types $A = B$ and $A \simeq B$ are equivalent, so we can instead show that $A \simeq B$ is a 0-type. By Lemma 2.120, the type of paths between equivalences $A \simeq B$ is equivalent to the type of paths between the underlying functions $A \to B$; hence, if $A \to B$ is a 0-type, so is $A \simeq B$. Since B is a finite type, it is a 0-type (Corollary 5.5), and hence $A \to B$ is also (Remark 2.75). □

Corollary 5.7. *For every $n : \mathbb{N}$, the type $\mathcal{BS}_n$ is a 1-type. Hence, $\mathcal{BS}_\bullet$ is a 1-type.*

Proof. Let $n : \mathbb{N}$. By Lemma 2.79, it is enough to show the following:

1. for every $A : \mathcal{U}$, the type $\|A \simeq [n]\|$ is a 1-type; and
2. for every $\langle A, t_A \rangle, \langle B, t_B \rangle : \mathcal{BS}_n$, the identity type $(A = B)$ is a 0-type.

The first condition is trivially met: $\|A \simeq [n]\|$ is a (-1)-type, and hence a 1-type. The second condition is proved in Lemma 5.6. □

We can immediately prove the following results about finite types.

Lemma 5.8. *$\mathcal{BS}_0$ and $\mathcal{BS}_1$ are contractible.*

Proof. The center of contraction of $\mathcal{BS}_0$ is $\langle [0], |\mathsf{id}_{[0]}| \rangle$. We then need to find a term in

$$\prod (X : \mathcal{BS}_0) . \langle [0], |\mathsf{id}_{[0]}| \rangle = X,$$

which (by the elimination principle of Σ-types) is obtained once found a term in

$$\prod (A : \mathcal{U}) . \prod (t : \|A \simeq [0]\|) . \langle [0], |\mathsf{id}_{[0]}| \rangle = \langle A, t \rangle.$$

By Remark 2.78, it is enough to provide a path $[0] = A$ in the base, under the assumption $t : \|A \simeq [0]\|$. Since $[0]$ is a (-1)-type (Remark 2.75), by Lemma 2.130 the type $[0] = A$ is also a (-1)-type, so by the elimination principle of the truncation we can work under the assumption $t' : A \simeq [0]$. Then $(\mathsf{ua}(t'))^{-1} : [0] = A$, so $\mathcal{BS}_0$ is contractible. Contractibility of $\mathcal{BS}_1$ is proved in the same way. □

Lemma 5.9. *For every $n : \mathbb{N}$, $\mathcal{BS}_n$ is connected.*

Proof. We want to show that $\|\mathcal{BS}_n\|_0$ is contractible for every n. The obvious choice for the center of the contraction is $|\langle [n], |\mathsf{id}_{[n]}| \rangle|$. Then, in order to show that

$$\prod \left(X : \|\mathcal{BS}_n\|_0 \right) . |\langle [n], |\mathsf{id}_{[n]}| \rangle| = X,$$

we can use the elimination principles of the propositional truncation and of Σ-types and, for every $A : \mathcal{U}$, find a term in

$$\prod (t : \|A \simeq [n]\|) . |\langle [n], |\mathsf{id}_{[n]}| \rangle| = |\langle A, t \rangle|.$$

Using again the elimination principle of the propositional truncation, such a function is obtained once provided a term in

$$\prod \left(t' : A \simeq [n] \right) . |\langle [n], |\mathsf{id}_{[n]}| \rangle| = |\langle A, |t'| \rangle|.$$

Given $t' : A \simeq [n]$, we have $(\mathsf{ua}(t'))^{-1} : [n] = A$, which is enough to give a path $\langle [n], |\mathsf{id}_{[n]}| \rangle = \langle A, |t'| \rangle$, by virtue of Remark 2.78. Applying $|-|$ to such a path proves the claim. □

We will now define a symmetric monoidal structure on the subuniverse of finite types. Since, by Corollary 5.7, $\mathcal{BS}_\bullet$ is a 1-type, conferring on it a symmetric monoidal structure will make it a symmetric monoidal groupoid. In order to simplify the exposition, we will often treat paths in $\mathcal{BS}_\bullet$ as paths in the equivalent type:

$$\Sigma\,(t : \mathbb{N} \times \mathcal{U})\,.\,\|\mathsf{pr}_2(t) \simeq [\mathsf{pr}_1(t)]\|\,.$$

The symmetric monoidal structure we define is the one induced by the one on $\mathcal{U}$ with the coproduct type former $+ : \mathcal{U} \to \mathcal{U} \to \mathcal{U}$ and the empty type $\mathbf{0}$ as unit, mentioned in Remark 4.2. There are families of equivalences

$$\begin{aligned}
\alpha^{\simeq}_{\mathcal{U}} &: \Pi\,(A, B, C : \mathcal{U})\,.\,A + B + C \simeq A + (B + C),\\
\lambda^{\simeq}_{\mathcal{U}} &: \Pi\,(B : \mathcal{U})\,.\,\mathbf{0} + B \simeq B,\\
\rho^{\simeq}_{\mathcal{U}} &: \Pi\,(A : \mathcal{U})\,.\,A + \mathbf{0} \simeq A,\\
\tau^{\simeq}_{\mathcal{U}} &: \Pi\,(A, B : \mathcal{U})\,.\,A + B \simeq B + A,
\end{aligned}$$

showing associativity, unitality and symmetry of the coproduct of types; these induce, via univalence, corresponding families of paths. For example, symmetry is the function term

$$\tau_{\mathcal{U}} :\equiv (A \mapsto B \mapsto \mathsf{ua}(\tau^{\simeq}_{\mathcal{U}}(A, B))) : \Pi\,(A, B : \mathcal{U})\,.\,A + B = B + A.$$

The 2-paths for the coherence diagram $\pentagon_{\mathcal{U}}$, $\triangledown_{\mathcal{U}}$, $\hexagon_{\mathcal{U}}$ and $\diamondsuit_{\mathcal{U}}$ are then found by examining their defining equivalences. For instance, the 2-path

$$\diamondsuit_{\mathcal{U}} : \Pi\,(A, B : \mathcal{U})\,.\,\tau_{\mathcal{U}}(A, B) \cdot \tau_{\mathcal{U}}(B, A) = \mathsf{refl}_{A+B}$$

is obtained, for $A, B : \mathcal{U}$, as the concatenation

$$\begin{aligned}
\tau_{\mathcal{U}}(A, B) \cdot \tau_{\mathcal{U}}(B, A) &\equiv \mathsf{ua}(\tau^{\simeq}_{\mathcal{U}}(A, B)) \cdot \mathsf{ua}(\tau^{\simeq}_{\mathcal{U}}(B, A))\\
&= \mathsf{ua}(\tau^{\simeq}_{\mathcal{U}}(B, A) \circ \tau^{\simeq}_{\mathcal{U}}(A, B)) && \text{by Lemma 2.113}\\
&= \mathsf{ua}(\mathsf{id}_{A+B})\\
&= \mathsf{refl}_{A+B} && \text{by Lemma 2.113,}
\end{aligned}$$

since there is an identity between the equivalences $\tau^{\simeq}_{\mathcal{U}}(B, A) \circ \tau^{\simeq}_{\mathcal{U}}(A, B)$ and id_{A+B}, as the underlying functions are homotopic (Lemma 2.120(iii)).

The type $\mathbb{N} : \mathcal{U}$ can also be endowed with a symmetric monoidal structure: addition in $\mathbb{N}$ is provably associative, symmetric (commutative) and unital with respect to $0 : \mathbb{N}$; the coherence diagrams are trivially satisfied, as $\mathbb{N}$ is a 0-type (Remark 2.75).

The symmetric monoidal structure on $\mathcal{B}\mathsf{S}_\bullet$ will rely on those on $\mathcal{U}$ and $\mathbb{N}$; we will only need to verify that finite types are closed under coproducts, and that this operation is graded on the cardinality.

Lemma 5.10. *The coproduct of two finite types is a finite type whose cardinality is the sum of the cardinalities of the summands; i.e., given types $A, B : \mathcal{U}$ and terms $n_A, n_B : \mathbb{N}$, $t_A : \|A \simeq [n_A]\|$ and $t_B : \|B \simeq [n_B]\|$, there is a term $t_{A+B} : \|(A + B) \simeq [n_A + n_B]\|$.*

Proof. Using the elimination principle of the truncation, it is enough to show that, given equivalences $e_A : A \simeq [n_A]$ and $e_B : B \simeq [n_B]$, we can construct an equivalence $e_{A+B} : (A + B) \simeq [n_A + n_B]$. This will be defined as the composition of $e_A + e_B : (A + B) \simeq ([n_A] + [n_B])$ with an equivalence $([n_A] + [n_B]) \simeq [n_A + n_B]$, which can be obtained by induction on n_A. □

Lemma 5.11. *The type $\mathcal{B}\mathsf{S}_\bullet$ has a symmetric monoidal structure, defined in the proof.*

Proof. The unit is given by $\underline{\mathbf{0}} :\equiv \langle 0, \mathbf{0}, |\mathsf{id}_{\mathbf{0}}| \rangle : \mathcal{B}\mathsf{S}_\bullet$. For a product $\boxplus : \mathcal{B}\mathsf{S}_\bullet \to \mathcal{B}\mathsf{S}_\bullet \to \mathcal{B}\mathsf{S}_\bullet$, by the elimination principle for Σ-types it is enough to define

$$\langle n_A, A, t_A \rangle \boxplus \langle n_B, B, t_B \rangle :\equiv \langle n_A + n_B, A + B, t_{A+B} \rangle$$

for every n_A, n_B, A, B t_A and t_B making the expression well-typed, with the term t_{A+B} constructed from t_A and t_B as in Lemma 5.10. The rest of the structure follows by combining the symmetric monoidal structures of $\mathbb{N}$ and $\mathcal{U}$, together with path algebra concerning Σ-types (in particular, applications of Remark 2.78). We have, for example,

$$\alpha_{\mathcal{B}\mathsf{S}_\bullet}(\langle n_A, A, t_A \rangle, \langle n_B, B, t_B \rangle, \langle n_C, C, t_C \rangle) :\equiv \langle \alpha_{\mathbb{N}}(n_A, n_B, n_C),\ \alpha_{\mathcal{U}}(A, B, C),\ \ldots \rangle,$$

where α_N is the associativity of addition in $\mathbb{N}$, and similarly for the other natural isomorphisms and coherence diagrams. □

We now want to construct a symmetric monoidal equivalence between $\mathsf{slist}(\mathbf{1})$ and $\mathcal{B}\mathsf{S}_\bullet$. In order to do so, we will use another symmetric monoidal groupoid as a stepping stone, representing the *deloopings* of symmetric groups (of any order). Theoretically, the notion of the delooping of a group G agrees with that of the classifying space of G; the difference here lies in the definition we provide, which makes

the construction of the loops explicit (and inductive). We discuss deloopings of symmetric groups in the following section.

Remark 5.12. We report that in [Fin+21] an equivalence is shown between the subuniverse of finite types ($\mathcal{B}\mathsf{S}_\bullet$ in the notation of this thesis) and a HIT $\mathbb{B}$ with the following constructors:

$$\begin{aligned}\mathbb{B} ::= {} & \mathsf{obj} : \mathbb{N} \to \mathbb{B} \mid \mathsf{hom} : \Pi\,(m, n : \mathbb{N})\,.\,([m] \simeq [n]) \to (\mathsf{obj}(m) = \mathsf{obj}(n)) \\ & \mid \mathsf{id{-}coh} : \Pi\,(n : \mathbb{N})\,.\,\mathsf{hom}_{n,n}(\mathsf{id}_{[n]}) = \mathsf{refl}_{\mathsf{obj}(n)} \\ & \mid \mathsf{comp{-}coh} : \Pi\,(n_1, n_2, n_3 : \mathbb{N})\,.\,\Pi\,(\alpha : [n_1] \simeq [n_2])\,.\,\Pi\,(\beta : [n_2] \simeq [n_3])\,. \\ & \qquad\qquad \mathsf{hom}_{n_1,n_3}(\beta \circ \alpha) = \mathsf{hom}_{n_1,n_2}(\alpha) \cdot \mathsf{hom}_{n_2,n_3}(\beta) \\ & \mid T_{\mathbb{B}} : \mathsf{IsHGpd}(\mathbb{B}).\end{aligned}$$

A function $\mathfrak{g} : \mathbb{B} \to \mathcal{B}\mathsf{S}_\bullet$ is constructed using the elimination principle of $\mathbb{B}$. This function computes, on the 0-constructor, $\mathfrak{g}(\mathsf{obj}(n)) \equiv \langle n, [n], |\mathsf{id}_{[n]}| \rangle$ for every $n : \mathbb{N}$, and it is defined such that

$$[\mathfrak{g}](\mathsf{hom}_{m,n}(e)) = \langle \ldots, \mathsf{ua}(e), \ldots \rangle,$$

where the first omitted term comes from a function $\Pi\,(m, n : \mathbb{N})\,.\,([m] \simeq [n]) \to (m = n)$, while the last term is obtained by the truncation level of $\|[m] \simeq [n]\|$. The function $\mathfrak{g}$ is then shown to be an equivalence, using an "encode-decode"-style proof. While providing a proof that the subuniverse of finite types is equivalent to a small type, this equivalence does not seem to be conducive to a proof that $\mathcal{B}\mathsf{S}_\bullet \simeq \mathsf{slist}(\mathbf{1})$, and hence to a proof that $\mathcal{B}\mathsf{S}_\bullet$ is a *free* symmetric monoidal groupoid in the sense of Chapter 4.

5.2 Deloopings of Symmetric Groups

We recall that, given a group G, its delooping is a groupoid with one point $*$ and a one-to-one correspondence between loops at $*$ and elements in G; the product of two elements in G is realized by the concatenation of the corresponding loops. Our aim is to define a type $\mathsf{del}_\bullet$ of deloopings of symmetric groups (of any degree), to be shown equivalent to the subuniverse $\mathcal{B}\mathsf{S}_\bullet$ of finite types, highlighting the combinatorial nature of finite types *in its elimination principle*.

Before giving the definition of $\mathsf{del}_\bullet$, we will examine the elements of the combinatorial structure of $\mathcal{B}\mathsf{S}_\bullet$ that we wish to replicate, in light of the fact that we should also be able to recognize the deloopings of the symmetric groups in $\mathcal{B}\mathsf{S}_\bullet$ itself:

- there is a finite type $\mathbf{0}$ of cardinality 0. The type $\mathbf{0} \simeq \mathbf{0}$ is contractible; this corresponds to the fact that S_0 is trivial;
- if A is a finite type of cardinality n, we can produce the type $A + \mathbf{1}$ of cardinality $n+1$; consequently, for every equivalence $e : A \simeq B$ of finite types, there is an equivalence $\mathsf{incr}(e) : A + \mathbf{1} \simeq B + \mathbf{1}$, with incr as in Definition 2.100. This corresponds to the inclusion $\mathsf{S}_n \hookrightarrow \mathsf{S}_{n+1}$;
- for every finite type A, there is a nontrivial automorphism

$$\omega_A : A + \mathbf{1} + \mathbf{1} \simeq A + \mathbf{1} + \mathbf{1}, \tag{5.13}$$

  which is the identity on A and swaps $\mathsf{inr}(*)$ with $\mathsf{inl}(\mathsf{inr}(*))$. This embodies the generator a_{n+1} of S_{n+2} in (4.10), i.e., the transposition of $n+1$ and $n+2$ in the finite set $\{1, \ldots, n+2\}$;
- the automorphism ω_A in (5.13) is provably an involution, i.e.

$$\omega_A \circ \omega_A = \mathsf{id}_{A+\mathbf{1}+\mathbf{1}}, \tag{5.14}$$

  and satisfies the following "braiding" relation (*à la* Yang-Baxter):

$$\omega_{A+\mathbf{1}} \circ \mathsf{incr}(\omega_A) \circ \omega_{A+\mathbf{1}} = \mathsf{incr}(\omega_A) \circ \omega_{A+\mathbf{1}} \circ \mathsf{incr}(\omega_A); \tag{5.15}$$

  the identity between equivalences derives from an easily verified homotopy between the underlying functions (Lemma 2.120(iii)) as shown in Fig. 5.1. Referring to (4.10), these two properties correspond, respectively, to the relations $a_{n+1}^2 = 1$ in the presentation of S_{n+2} and $a_{n+2}a_{n+1}a_{n+2} = a_{n+1}a_{n+2}a_{n+1}$ in the presentation of S_{n+3};
- moreover, $\mathcal{B}\mathsf{S}_n$ is a 1-type.

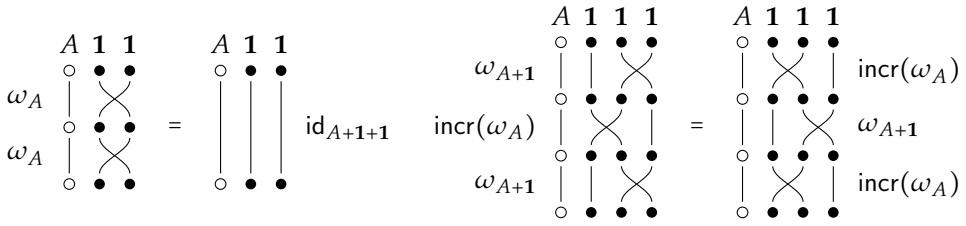


Figure 5.1: Combinatorial structure of the subuniverse of finite types, in pictures: ω_A is an involution (left) and it "braids" with $\omega_{A+\mathbf{1}}$ (right).

The combinatorial structure of $\mathcal{B}\mathsf{S}_\bullet$ presented above concerns equivalences of finite types. Via univalence, the same structure translates to one involving *paths*

between finite types; this suggests the use of HITs to define $\mathsf{del}_\bullet$, modelling finite types as terms, and paths between finite types as paths between the corresponding terms. The implicit claim, which will motivate our choice for the constructors of $\mathsf{del}_\bullet$, is that the combinatorial structure described above *generates* all paths in $\mathcal{B}\mathsf{S}_\bullet$, and hence it lends itself to an inductive definition. For practical reasons, we found it convenient to define $\mathsf{del}_\bullet$ "degreewise", i.e. uniformly as a family $\mathsf{del} : \mathbb{N} \to \mathcal{U}$ of types indexed by the natural numbers (as for $\mathcal{B}\mathsf{S}$ in Definition 5.3), each type in the family representing the delooping of the symmetric group of a certain degree.

Remark 5.16 (Notation). For $A : \mathcal{U}$, we will use the symbol

$$\gamma_A :\equiv \mathsf{ua}(\omega_A) : A + \mathbf{1} + \mathbf{1} = A + \mathbf{1} + \mathbf{1}, \tag{5.17}$$

with ω_A defined in (5.13). It follows that $\omega_A = \mathsf{ua}^{-1}(\gamma_A)$. From Lemma 2.113 and (5.14), we obtain a 2-path $\gamma_A^{-1} \equiv \mathsf{ua}(\omega_A)^{-1} = \mathsf{ua}(\omega_A^{-1}) = \mathsf{ua}(\omega_A) \equiv \gamma_A$.

Definition 5.18 (Deloopings of symmetric groups)**.** The ap-recursive, indexed family $\mathsf{del} : \mathbb{N} \to \mathcal{U}$ of 1-truncated HITs is defined with the following presentation:[2]

$$\begin{aligned}
\mathsf{del}_{(-)} ::= {} & \mathsf{pt}_0 : \mathsf{del}_0 \\
| \ & \mathsf{i} : \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \to \mathsf{del}_{n+1} \\
| \ & \mathsf{tw} : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : \mathsf{del}_n)\,.\,\mathsf{i}_{n+1}(\mathsf{i}_n(a)) = \mathsf{i}_{n+1}(\mathsf{i}_n(a)) \\
| \ & \mathsf{do} : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : \mathsf{del}_n)\,.\,\mathsf{tw}_n(a) \cdot \mathsf{tw}_n(a) = \mathsf{refl}_{\mathsf{i}_{n+1}(\mathsf{i}_n(a))} \\
| \ & \mathsf{br} : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : \mathsf{del}_n)\,.\,\mathsf{tw}_{n+1}(\mathsf{i}_n(a)) \cdot [\mathsf{i}_{n+2}](\mathsf{tw}_n(a)) \cdot \mathsf{tw}_{n+1}(\mathsf{i}_n(a)) \\
& \qquad\qquad = [\mathsf{i}_{n+2}](\mathsf{tw}_n(a)) \cdot \mathsf{tw}_{n+1}(\mathsf{i}_n(a)) \cdot [\mathsf{i}_{n+2}](\mathsf{tw}_n(a)) \\
| \ & T_{\mathsf{del}} : \Pi\,(n : \mathbb{N})\,.\,\mathsf{IsHGpd}(\mathsf{del}_n).
\end{aligned}$$

We will use the notation

$$\mathsf{del}_\bullet :\equiv \Sigma\,(n : \mathbb{N})\,.\,\mathsf{del}_n$$

for the coproduct, which is a 1-type (Remark 2.75). The index n in i, tw, do and br will be always omitted in the notation, and we will write tw_a for $\mathsf{tw}(a)$. The family is depicted in Fig. 5.2.

The elimination principle of $\mathsf{del}_{(-)}$ will treat the family uniformly in the degree n, and hence will pertain eliminating into $\mathbb{N}$-indexed families of *families* of types. Explicitly, given $C : \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \to \mathcal{U}$, in order to obtain a section

$$\mathsf{ind}_{\mathsf{del}} : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : \mathsf{del}_n)\,.\,C_n(a),$$

[2]The names for the constructors were chosen with pt_0 standing for "point"; i for "inclusion"; tw for "twist"; do for "double"; br for "braid".

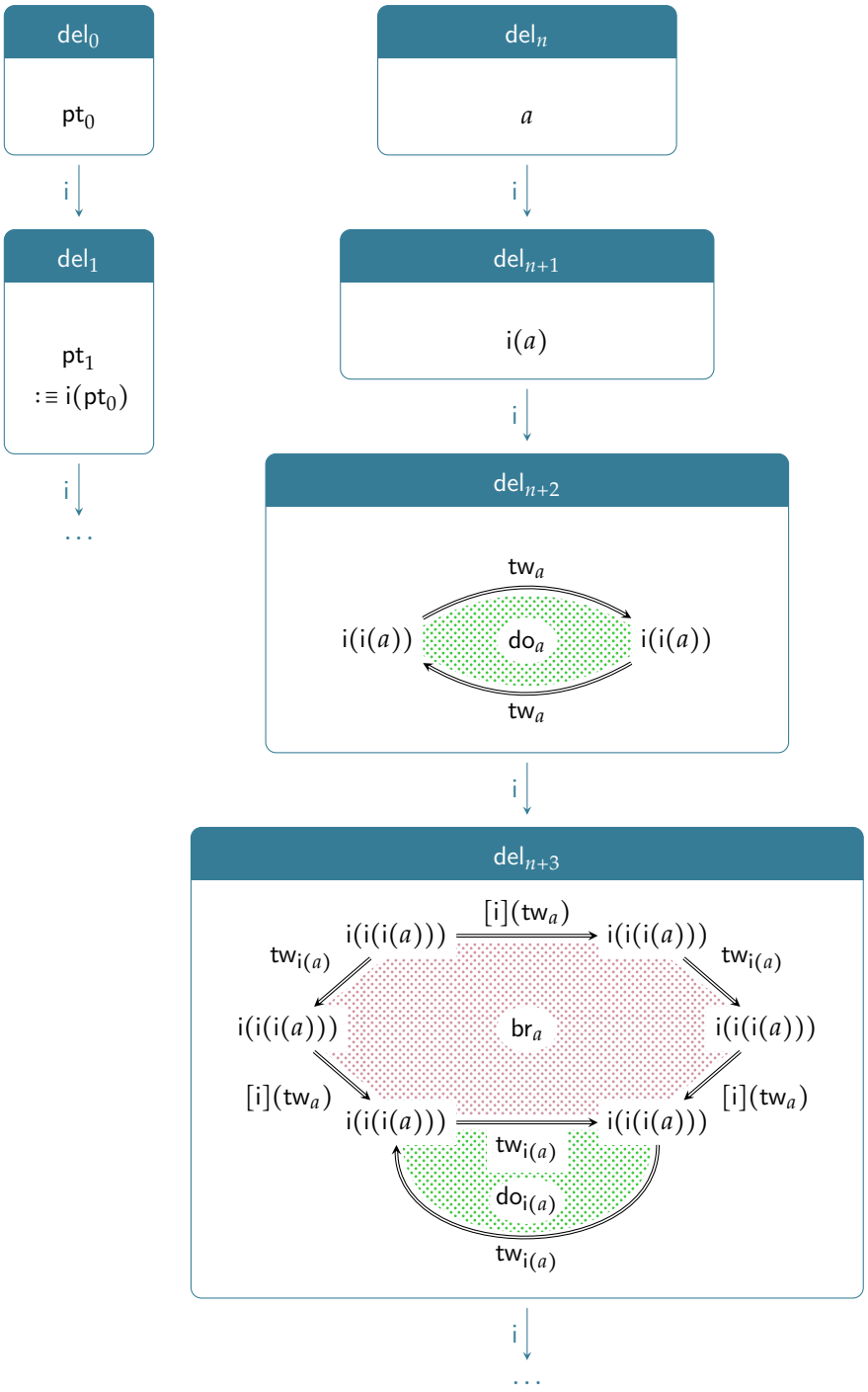


Figure 5.2: The indexed family del : $\mathbb{N} \to \mathcal{U}$ of HITs; some of the defining loops and 2-paths are shown for $a : \mathsf{del}_n$, for some $n : \mathbb{N}$. As we will see, the 0-truncation of del_n is contractible for every $n : \mathbb{N}$ (Lemma 5.26).

we need:

- a term $\mathsf{pt}' : C_0(\mathsf{pt}_0)$;
- a term $\mathsf{i}' : \prod(n : \mathbb{N}) . \prod(a : \mathsf{del}_n) . C_n(a) \to C_{n+1}(\mathsf{i}(a))$;
- a family of pathovers

$$\mathsf{tw}' : \prod(n : \mathbb{N}) . \prod(a : \mathsf{del}_n) . \prod\left(a' : C_n(a)\right) . (\mathsf{tw}_a)_*^{C_{n+2}}(\mathsf{i}'(\mathsf{i}'(a'))) = \mathsf{i}'(\mathsf{i}'(a')),$$

  where the first two arguments of i' are left implicit;

and so on, following the induction scheme presented in Section 2.6. The non-dependent version of this elimination principle applies to families of types indexed by $\mathbb{N}$; for $C : \mathbb{N} \to \mathcal{U}$, a term $\mathsf{rec}_{\mathsf{del}} : \prod(n : \mathbb{N}) . \mathsf{del}_n \to C_n$ is provided once given terms for C directly corresponding to the constructors of $\mathsf{del}_{(-)}$.

Definition 5.19. All types in the family del are inhabited; we define

$$\mathsf{pt} : \prod(n : \mathbb{N}) . \mathsf{del}_n$$

by induction on n, setting $\mathsf{pt}(0) :\equiv \mathsf{pt}_0$ and, recursively, $\mathsf{pt}(n+1) :\equiv \mathsf{i}(\mathsf{pt}(n))$. We will write $\mathsf{pt}_n :\equiv \mathsf{pt}(n)$ for every $n : \mathbb{N}$, where the judgmental equality resolves the notational ambiguity of pt_0 being also used to denote a constructor of $\mathsf{del}_{(-)}$.

Remark 5.20. The definition of $\mathsf{del}_{(-)}$ offers a counterpart to the combinatorial structure of $\mathcal{BS}_\bullet$ described at the beginning of this section, which, as we discussed, appears in the classical presentation of the symmetric groups S_n in (4.10). The generators a_i correspond to transpositions of adjacent elements in the finite set $\{1, \dots, n\}$; these are embodied by $\omega_{(-)}$ in $\mathcal{BS}_\bullet$ and by $\mathsf{tw}_{(-)}$ in $\mathsf{del}_\bullet$. As for the relations, both $a_i^2 = 1$ and $a_{i+1}a_ia_{i+1} = a_ia_{i+1}a_i$ are found in the specified combinatorial structure of finite sets and in the constructors of $\mathsf{del}_{(-)}$ (respectively, $\mathsf{do}_{(-)}$ and $\mathsf{br}_{(-)}$), while the relation $a_ia_j = a_ja_i$ for $|i - j| \geq 2$ is redundant in $\mathcal{BS}_\bullet$ and $\mathsf{del}_\bullet$, as shown in the following lemma (see also [ML98, Chapter XI, proof of Theorem 1] and Lemma 4.12 for related instances of the same phenomenon).

Lemma 5.21 (Naturality of tw, γ and ω)**.** *Recall* add *and* incr *from Definition 2.42 and Definition 2.100. With regard to* $\mathsf{del}_\bullet$ *and* $\mathcal{U}$ *(hence* $\mathcal{BS}_\bullet$*), the following holds:*

(i) the loops $\mathsf{tw}_{(-)}$ *commute with the concatenation with* $[\mathsf{i}]([\mathsf{i}](-))$*, i.e., for every* $n : \mathbb{N}$*,* $a, b : \mathsf{del}_n$ *and* $p : a = b$*, there is a* 2*-path*

$$[\mathsf{i}]([\mathsf{i}](p)) \cdot \mathsf{tw}_b = \mathsf{tw}_a \cdot [\mathsf{i}]([\mathsf{i}](p)); \tag{5.22}$$

(ii) the loops $\gamma_{(-)}$ commute with the concatenation with $[\mathsf{add}]([\mathsf{add}](-))$*, i.e., for every $A, B : \mathcal{U}$ and $p : A = B$, there is a 2-path*

$$[\mathsf{add}]([\mathsf{add}](p)) \cdot \gamma_B = \gamma_A \cdot [\mathsf{add}]([\mathsf{add}](p)); \tag{5.23}$$

(iii) the symmetries $\omega_{(-)}$ commute with the composition with $\mathsf{incr}(\mathsf{incr}(-))$*, i.e., for every $A, B : \mathcal{U}$ and $e : A \simeq B$, there is an identity between equivalences*

$$\omega_B \circ \mathsf{incr}(\mathsf{incr}(e)) = \mathsf{incr}(\mathsf{incr}(e)) \circ \omega_A, \tag{5.24}$$

where, we remind, $\mathsf{incr}(e) : \mathsf{add}(A) \simeq \mathsf{add}(B)$.

Proof. The first two claims are proved by induction on p, while (5.24) can be derived, for example, from the second one; indeed:

$$\begin{aligned}
\omega_B \circ \mathsf{incr}(\mathsf{incr}(e)) &= \mathsf{ua}^{-1}(\gamma_B) \circ (\mathsf{incr}(\mathsf{incr}(\mathsf{ua}^{-1}(\mathsf{ua}(e))))) \\
&= \mathsf{ua}^{-1}(\gamma_B) \circ \mathsf{ua}^{-1}([\mathsf{add}]([\mathsf{add}](\mathsf{ua}(e)))) && \text{by (2.129)} \\
&= \mathsf{ua}^{-1}([\mathsf{add}]([\mathsf{add}](p)) \cdot \gamma_B) && \text{by path algebra} \\
&= \mathsf{ua}^{-1}(\gamma_A \cdot [\mathsf{add}]([\mathsf{add}](p))) && \text{by (5.23)} \\
&= \mathsf{incr}(\mathsf{incr}(e)) \circ \omega_A && \text{in reverse.}
\end{aligned}$$

Alternatively, an identity between the two equivalences is obtained because the underlying functions $A \to B$ are homotopic (Lemma 2.120(iii)), which can be easily verified by induction on any argument in $A + \mathbf{1} + \mathbf{1}$. □

The results about $\mathcal{B}\mathsf{S}_\bullet$ shown in Section 5.1 have a counterpart in $\mathsf{del}_\bullet$.

Lemma 5.25. *The types* del_0 *and* del_1 *are contractible.*

Proof. We will start by proving that del_0 is contractible. The chosen center of contraction is, obviously, pt_0; we will thus need to find a term $\mathsf{contr}_0 : \Pi\,(a : \mathsf{del}_0)\,.\,\mathsf{pt}_0 = a$ as proof of contraction. The elimination principle of $\mathsf{del}_{(-)}$ cannot be used directly to produce a dependent function out of del_0, as it applies to families indexed over the natural numbers (recall Remark 2.44). For this reason, we need to generalize our goal to finding a term

$$\mathsf{contr}'_0 : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : \mathsf{del}_n)\,.\,(n = 0) \to (\mathsf{pt}_n = a),$$

so that we can apply the elimination principle of $\mathsf{del}_{(-)}$ to the (indexed) family $C : \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \to \mathcal{U}$ defined by $C_n(a) :\equiv (n = 0) \to (\mathsf{pt}_n = a)$. By function extensionality, every type in the family C is a 0-type, because the target of the function is an identity type in a 1-type. Hence, such a function can be obtained once provided:

- a term $\mathsf{pt}'_0 : (0 = 0) \to (\mathsf{pt}_0 = \mathsf{pt}_0)$, which is trivially given as the function constant at the identity path;
- for every $n : \mathbb{N}$ and $a : \mathsf{del}_n$, a term

$$\mathsf{i}'_n(a) : ((n = 0) \to (\mathsf{pt}_n = a)) \to (n + 1 = 0) \to (\mathsf{pt}_{n+1} = \mathsf{i}(a));$$

  this can be obtained *ex falso*, as there is a function $\Pi\,(n : \mathbb{N})\,.\,(n + 1 = 0) \to \mathbf{0}$;
- for every $n : \mathbb{N}$, $a : \mathsf{del}_n$ and $a' : (n = 0) \to (\mathsf{pt}_n = a)$, a pathover

$$\mathsf{tw}'_n(a) : (\mathsf{tw}_a)^{C_{n+2}}_*(\mathsf{i}'_{n+1}(\mathsf{i}'_n(a'))) = \mathsf{i}'_{n+1}(\mathsf{i}'_n(a'));$$

  by function extensionality, it is sufficient to show that the two functions agree on all $p : n + 2 = 0$; again, this is proved *ex falso*.

The proof of contraction is then achieved by defining $\mathsf{contr}_0(a) :\equiv \mathsf{contr}'_0(0, a, \mathsf{refl}_0)$.

Contractibility of del_1 is proved similarly: the center of contraction, in this case, is pt_1, and again we use the elimination principle of $\mathsf{del}_{(-)}$ to find a term

$$\mathsf{contr}'_1 : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : \mathsf{del}_n)\,.\,(n = 1) \to (\mathsf{pt}_n = a).$$

We need:

- a term $\mathsf{pt}'_0 : (0 = 1) \to (\mathsf{pt}_0 = \mathsf{pt}_0)$, trivially given;
- for every $n : \mathbb{N}$ and $a : \mathsf{del}_n$, a term

$$\mathsf{i}'_n(a) : ((n = 1) \to (\mathsf{pt}_n = a)) \to (n + 1 = 1) \to (\mathsf{pt}_{n+1} = \mathsf{i}(a)).$$

  The function i'_n will be, once again, constant on the first argument (the term in $(n = 1) \to (\mathsf{pt}_n = a)$). By induction on n:
  - for $a : \mathsf{del}_0$, a term in $(1 = 1) \to (\mathsf{pt}_1 = \mathsf{i}(a))$, is given by the function $(p \mapsto [\mathsf{i}](\mathsf{contr}_0(a)))$, since $\mathsf{pt}_1 \equiv \mathsf{i}(\mathsf{pt}_0)$;
  - the inductive case seeks, for $a : \mathsf{del}_{n+1}$ and some inductive hypothesis, a term in $(n + 2 = 1) \to (\mathsf{pt}_{n+2} = \mathsf{i}(a))$, which is given *ex falso*;
- the requirement tw' is resolved in the same way as for the proof of contractibility of del_0. □

The same argument fails for del_2, and in general for an arbitrary $n : \mathbb{N}$ other than 0 or 1. Indeed, assuming that del_2 is contractible would imply the existence of a path $\mathsf{tw}_{\mathsf{pt}_0} = \mathsf{refl}_{\mathsf{pt}_2}$, which leads to a contradiction, as we will see in Lemma 5.46. Conversely, assuming a 2-path $p : \mathsf{tw}_a = \mathsf{refl}_{\mathsf{i}(\mathsf{i}(a))}$ for every $n : \mathbb{N}$ and $a : \mathsf{del}_n$ is

enough to show contractibility of all types in the family del. Hence, it does not come as a surprise that this result can be achieved under the stronger assumption of having *all* 2-paths.

Lemma 5.26. *For every $n : \mathbb{N}$, del_n is connected.*

Proof. We want to show that $\|\mathsf{del}_n\|_0$ is contractible for every n. Using $|\mathsf{pt}_n|$ as center of the contraction, we need to find a term in $\prod(n : \mathbb{N}) . \prod\left(a : \|\mathsf{del}_n\|_0\right) . |\mathsf{pt}_n| = a$. By the elimination principle of the truncation, it is enough to find a term in

$$\prod(n : \mathbb{N}) . \prod(a : \mathsf{del}_n) . |\mathsf{pt}_n| = |a|,$$

which we can do using the elimination principle of $\mathsf{del}_{(-)}$, where we eliminate into an indexed family of (-1)-types. Therefore, it suffices to provide:

- a path $\mathsf{pt}'_0 : |\mathsf{pt}_0| = |\mathsf{pt}_0|$, given by the identity path;
- for every $n : \mathbb{N}$, $a : \mathsf{del}_n$ and $a' : |\mathsf{pt}_n| = |a|$, a path $\mathsf{i}'_n(a, a') : |\mathsf{i}(\mathsf{pt}_n)| = |\mathsf{i}(a)|$. The function $\mathsf{i} : \mathsf{del}_n \to \mathsf{del}_{n+1}$ induces a map $\|\mathsf{i}\|_0 : \|\mathsf{del}_n\|_0 \to \|\mathsf{del}_{n+1}\|_0$ computing $\|\mathsf{i}\|_0 (|x|) \equiv |\mathsf{i}(x)|$ for every $x : \mathsf{del}_n$, so $[\|\mathsf{i}\|_0](a')$ is the sought path. □

Using connectedness, we can prove properties (in the sense of Definition 2.73) of terms in any of the types of the family del, without applying the elimination principle of the HIT. The following lemma provides an example of the application of this proof technique.

Lemma 5.27. *Let $n : \mathbb{N}$. For every $a : \mathsf{del}_{n+1}$, there is a term in $\|\sum(x : \mathsf{del}_n) . a = \mathsf{i}(x)\|$.*

Proof. Since del_n is connected (Lemma 5.26), by Lemma 2.143 it is enough to find a term in the type $\|\sum(x : \mathsf{del}_n) . \mathsf{pt}_{n+1} = \mathsf{i}(x)\|$; this can be given by $|\langle \mathsf{pt}_n, \mathsf{refl}_{\mathsf{pt}_{n+1}} \rangle|$. □

Like $\mathcal{B}\mathsf{S}_\bullet$, also $\mathsf{del}_\bullet \equiv \sum(n : \mathbb{N}) . \mathsf{del}_n$ can be endowed with a symmetric monoidal structure; this will require multiple applications of the elimination principle of $\mathsf{del}_{(-)}$. We will present its most prominent features, while leaving out some details. In the Coq formalization, the symmetric monoidal structure is defined almost completely, save for the the coherence diagrams ⎔ and ⟲; these can be proved similarly to the ones for $\mathsf{slist}(X)$ (as we will see in Lemma 5.30).

We will start by presenting the symmetric monoidal product $\oplus : \mathsf{del}_\bullet \to \mathsf{del}_\bullet \to \mathsf{del}_\bullet$. The specifics of its definition are motivated by the fact that, ultimately, we would like to construct a symmetric monoidal equivalence $\mathsf{del}_\bullet \simeq \mathcal{B}\mathsf{S}_\bullet$ by which, if $a : \mathsf{del}_n$ corresponds to a finite type A, then $\mathsf{i}(a) : \mathsf{del}_{n+1}$ should correspond to $A + \mathbf{1}$. In particular, referring to the definition of the symmetric monoidal product $\boxplus$ in

$\mathcal{BS}_\bullet$ (Lemma 5.11), we want to define $\oplus$ so that $\mathfrak{f}_\bullet(x) \boxplus \mathfrak{f}_\bullet(y) = \mathfrak{f}_\bullet(x \oplus y)$ for every x, $y : \mathsf{del}_\bullet$. Since, on the base of the Σ-type, the product $\boxplus$ coincides with the addition on natural numbers, the product $\oplus$ will too; that is, the type

$$\Pi\,(a, b : \mathsf{del}_\bullet)\,.\,\mathsf{pr}_1(a \oplus b) = \mathsf{pr}_1(a) + \mathsf{pr}_1(b)$$

will be inhabited (we omit the proof, which is straightforward).

Definition 5.28. A function $\oplus : \mathsf{del}_\bullet \to \mathsf{del}_\bullet \to \mathsf{del}_\bullet$ is defined, using the elimination principle of Σ-types, by a function

$$\Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : \mathsf{del}_n)\,.\,\mathsf{del}_\bullet \to \mathsf{del}_\bullet,$$

which, in turn, is obtained by the elimination principle of $\mathsf{del}_{(-)}$, as follows.

- A term $\mathsf{pt}'_0 : \mathsf{del}_\bullet \to \mathsf{del}_\bullet$ is given by the identity function;
- for every $n : \mathbb{N}$, $a : \mathsf{del}_n$ and $a' : \mathsf{del}_\bullet \to \mathsf{del}_\bullet$, a function $\mathsf{i}'(a') : \mathsf{del}_\bullet \to \mathsf{del}_\bullet$ is defined by $\mathsf{i}'(a') :\equiv \langle \mathsf{s}, \mathsf{i} \rangle \circ a'$, using the notation of Definition 2.18, where, we remind, s is the successor function in $\mathbb{N}$;
- for every $n : \mathbb{N}$, $a : \mathsf{del}_n$ and $a' : \mathsf{del}_\bullet \to \mathsf{del}_\bullet$, a path

  $$\mathsf{tw}' : \langle \mathsf{s}, \mathsf{i} \rangle \circ \langle \mathsf{s}, \mathsf{i} \rangle \circ a' = \langle \mathsf{s}, \mathsf{i} \rangle \circ \langle \mathsf{s}, \mathsf{i} \rangle \circ a'$$

  is obtained by function extensionality; for every $x : \mathsf{del}_\bullet$, we need to find a loop at $\langle \mathsf{s}, \mathsf{i} \rangle (\langle \mathsf{s}, \mathsf{i} \rangle (a'(x))) : \mathsf{del}_\bullet$, i.e., a path

  $$\langle \mathsf{pr}_1(a'(x)) + 2,\ \mathsf{i}(\mathsf{i}(\mathsf{pr}_2(a'(x))))\rangle = \langle \mathsf{pr}_1(a'(x)) + 2,\ \mathsf{i}(\mathsf{i}(\mathsf{pr}_2(a'(x))))\rangle;$$

  this is given by $\langle \mathsf{refl}_{\mathsf{pr}_1(a'(x))+2},\ \mathsf{tw}_{\mathsf{pr}_2(a'(x))} \rangle$;
- the families of 2-paths relative to the constructors do and br are obtained similarly, using properties of function extensionality, path algebra, and the constructors do and br themselves; for example, for $n : \mathbb{N}$, $a : \mathsf{del}_n$ and $a' : \mathsf{del}_\bullet \to \mathsf{del}_\bullet$, a 2-path $\mathsf{do}' : \mathsf{tw}' \cdot \mathsf{tw}' = \mathsf{refl}_{(\mathsf{i}'(\mathsf{i}'(a')) = \mathsf{i}'(\mathsf{i}'(a')))}$ is constructed, for every $x : \mathsf{del}_\bullet$, as the concatenation

  $$\begin{aligned}
  &\langle \mathsf{refl}_{\mathsf{pr}_1(a'(x))},\ \mathsf{tw}_{\mathsf{pr}_2(a'(x))} \rangle \cdot \langle \mathsf{refl}_{\mathsf{pr}_1(a'(x))},\ \mathsf{tw}_{\mathsf{pr}_2(a'(x))} \rangle \\
  &= \langle \mathsf{refl}_{\mathsf{pr}_1(a'(x))},\ \mathsf{tw}_{\mathsf{pr}_2(a'(x))} \cdot^{\mathsf{d}} \mathsf{tw}_{\mathsf{pr}_2(a'(x))} \rangle \\
  &\equiv \langle \mathsf{refl}_{\mathsf{pr}_1(a'(x))},\ \mathsf{tw}_{\mathsf{pr}_2(a'(x))} \cdot \mathsf{tw}_{\mathsf{pr}_2(a'(x))} \rangle \\
  &= \langle \mathsf{refl}_{\mathsf{pr}_1(a'(x))},\ \mathsf{refl}_{\mathsf{pr}_2(a'(x))} \rangle && \text{by } \mathsf{do}_{\mathsf{pr}_2(a'(x))} \\
  &\equiv \mathsf{refl}_{a'(x)},
  \end{aligned}$$

using the notation of Lemma 2.69. The function $\oplus$ then enjoys the computational properties:

$$\langle 0, \mathsf{pt}_0 \rangle \oplus x \equiv x, \qquad \langle n+1, \mathsf{i}(a) \rangle \oplus x \equiv \langle \mathsf{s}, \mathsf{i} \rangle (\langle n, a \rangle \oplus x), \tag{5.29}$$

for every $n : \mathbb{N}$, $a : \mathsf{del}_n$ and $x : \mathsf{del}_\bullet$.

Lemma 5.30. *The type* $\mathsf{del}_\bullet$ *has a symmetric monoidal structure, defined in the proof.*

Proof. The symmetric monoidal structure we define on $\mathsf{del}_\bullet$ has the term $\underline{\mathsf{pt}_0} :\equiv \langle 0, \mathsf{pt}_0 \rangle : \mathsf{del}_\bullet$ as unit and the function $\oplus : \mathsf{del}_\bullet \to \mathsf{del}_\bullet \to \mathsf{del}_\bullet$ in Definition 5.28 as symmetric monoidal product. The natural isomorphisms $\alpha_{\mathsf{del}_\bullet}$, $\rho_{\mathsf{del}_\bullet}$ and $\tau_{\mathsf{del}_\bullet}$ defining the symmetric monoidal structure are obtained by the elimination principle of $\mathsf{del}_{(-)}$, while $\lambda_{\mathsf{del}_\bullet}$ holds judgmentally by (5.29). Starting from associativity, for a family of paths $\alpha_{\mathsf{del}_\bullet} : \Pi\,(x, y, z : \mathsf{del}_\bullet)\,.\,(x \oplus y) \oplus z = x \oplus (y \oplus z)$ we need:

- a family of paths $(\langle 0, \mathsf{pt}_0 \rangle \oplus y) \oplus z = \langle 0, \mathsf{pt}_0 \rangle \oplus (y \oplus z)$, all given by the reflexivity path, since both sides of the identity type compute to $y \oplus z$;
- given $n : \mathbb{N}$, $a : \mathsf{del}_n$ and $y, z : \mathsf{del}_\bullet$, and assuming inductively a path

  $$h : (\langle n, a \rangle \oplus y) \oplus z = \langle n, a \rangle \oplus (y \oplus z), \tag{5.31}$$

  a path $(\langle n+1, \mathsf{i}(a) \rangle \oplus y) \oplus z = \langle n+1, \mathsf{i}(a) \rangle \oplus (y \oplus z)$ is found directly:

  $$\begin{aligned}
  (\langle n+1, \mathsf{i}(a) \rangle \oplus y) \oplus z &\equiv (\langle \mathsf{s}, \mathsf{i} \rangle (\langle n, a \rangle \oplus y)) \oplus z \\
  &\equiv \langle \mathsf{pr}_1(\langle n, a \rangle \oplus y) + 1,\ \mathsf{i}(\mathsf{pr}_2(\langle n, a \rangle \oplus y)) \rangle \oplus z \\
  &\equiv \langle \mathsf{s}, \mathsf{i} \rangle ((\langle n, a \rangle \oplus y) \oplus z) \\
  &= \langle \mathsf{s}, \mathsf{i} \rangle (\langle n, a \rangle \oplus (y \oplus z)) \qquad \text{by } [\langle \mathsf{s}, \mathsf{i} \rangle](h) \\
  &\equiv \langle n+1, \mathsf{i}(a) \rangle \oplus (y \oplus z);
  \end{aligned}$$

- given $n : \mathbb{N}$, $a : \mathsf{del}_n$, $y, z : \mathsf{del}_\bullet$ and h as in (5.31), the requirement relative to the constructor tw of $\mathsf{del}_{(-)}$ is fulfilled by the 2-path shown in Fig. 5.3.

A family $\rho_{\mathsf{del}_\bullet}$ of paths for right unitality is obtained similarly. About symmetry, defining a family $\tau_{\mathsf{del}_\bullet} : \Pi\,(x, y : \mathsf{del}_\bullet)\,.\,x \oplus y = y \oplus x$ requires a nested application of the elimination principle of $\mathsf{del}_{(-)}$. Its construction follows closely the definition of τ_{slist} for $+\!\!+$ in Lemma 4.29. A family of paths

$$Q_y : \Pi\,(x : \mathsf{del}_\bullet)\,.\,\langle \mathsf{s}, \mathsf{i} \rangle (x \oplus y) = x \oplus \langle \mathsf{s}, \mathsf{i} \rangle (y)$$

is defined for every $y : \mathsf{del}_\bullet$ by the elimination principle of $\mathsf{del}_\bullet$, computing

$$Q_y \langle 0, \mathsf{pt}_0 \rangle \equiv \mathsf{refl}_{\langle \mathsf{s}, \mathsf{i} \rangle (y)}, \qquad Q_y \langle n+1, \mathsf{i}(a) \rangle \equiv \langle \mathsf{refl}, \mathsf{tw}_{\mathsf{pr}_2(\langle n, a \rangle \oplus y)} \rangle \cdot [\langle \mathsf{s}, \mathsf{i} \rangle](Q_y \langle n, a \rangle),$$

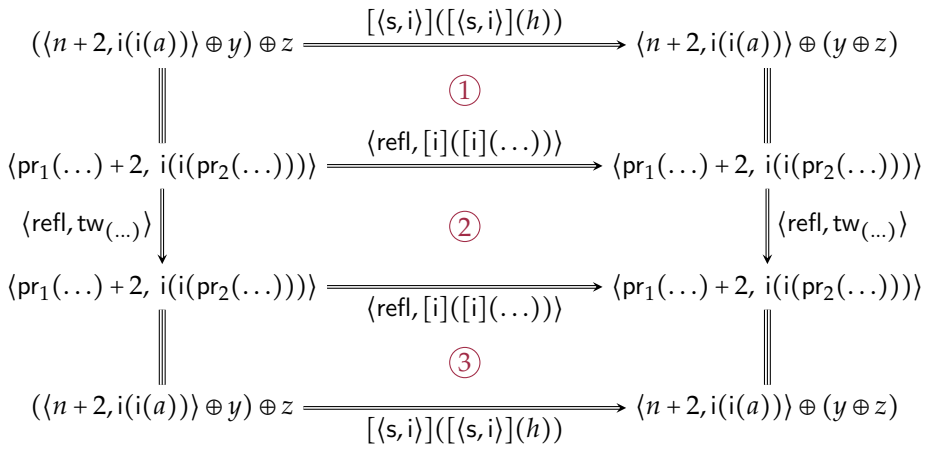


Figure 5.3: Requirement for the constructor tw in the definition of $\alpha_{\mathsf{del}_\bullet}$. The 2-paths in (1) and (3) are given by (2.67), together with the fact that $\mathbb{N}$ is a 0-type (so there is a 2-path between the path in the base and refl); (2) is given by Lemma 5.21.

for every $n : \mathbb{N}$ and $a : \mathsf{del}_n$ (cf. Lemma 4.25 and relevant Fig. 4.16 for the requirement relative to the 1-constructor), and such that there is a family of 2-paths

$$\begin{aligned} R_{x,y} : {} & [\langle \mathsf{s}, \mathsf{i}\rangle](Q_y(x)) \cdot Q_{\langle \mathsf{s},\mathsf{i}\rangle(y)}(x) \cdot (\mathsf{refl}_x \oplus \langle \mathsf{refl}_{\mathsf{pr}_1(y)+2}, \mathsf{tw}_{\mathsf{pr}_2(y)}\rangle) \\ & = (\langle \mathsf{refl}_{\mathsf{pr}_1(x)+2}, \mathsf{tw}_{\mathsf{pr}_2(x)}\rangle \oplus y) \cdot [\langle \mathsf{s}, \mathsf{i}\rangle](Q_y(x)) \cdot Q_{\langle \mathsf{s},\mathsf{i}\rangle(y)}(x) \end{aligned}$$

for every $x, y : \mathsf{del}_\bullet$ (cf. Lemma 4.28). Then $\tau_{\mathsf{del}_\bullet}$ can be defined by

$$\begin{aligned} \tau_{\mathsf{del}_\bullet}(\langle 0, \mathsf{pt}_0\rangle, y) :\equiv \rho^{-1}_{\mathsf{del}_\bullet}(y) \qquad & : \langle 0, \mathsf{pt}_0\rangle \oplus y = y \oplus \langle 0, \mathsf{pt}_0\rangle, \\ \tau_{\mathsf{del}_\bullet}(\langle n+1, \mathsf{i}(a)\rangle, y) :\equiv [\langle \mathsf{s}, \mathsf{i}\rangle(\tau_{\mathsf{del}_\bullet}(\langle n, a\rangle, y))] \cdot Q_{\langle n,a\rangle}(y) & \\ : \langle n+1, \mathsf{i}(a)\rangle \oplus y = y \oplus \langle n+1, \mathsf{i}(a)\rangle, & \end{aligned}$$

for every $n : \mathbb{N}$, $a : \mathsf{del}_n$ and $y : \mathsf{del}_\bullet$, and via a 2-path relative to the 1-constructor tw obtained via the family R in a way similar to the one displayed in Fig. 4.17 for τ_{slist}.

The families $\pentagon_{\mathsf{del}_\bullet}$, $\triangledown_{\mathsf{del}_\bullet}$, $\hexagon_{\mathsf{del}_\bullet}$ and $\diamondsuit_{\mathsf{del}_\bullet}$ of 2-paths encoding the coherence diagrams require only induction at the level of the 0-constructors pt_0 and i of $\mathsf{del}_{(-)}$; these are found, again, similarly to their counterparts for slist. □

5.3 An Equivalence $\mathsf{slist}(\mathbf{1}) \simeq \mathsf{del}_\bullet$

This section is devoted to the construction of a symmetric monoidal equivalence $\mathsf{slist}(\mathbf{1}) \simeq \mathsf{del}_\bullet$. It is easy to imagine how the symmetric monoidal functors in the two

directions should be defined: indeed, the constructors of the two higher inductive types are very similar. The main difference between the two definitions is that $\mathsf{del}_{(-)}$ is indexed by the natural numbers; each member del_n of the family should then correspond to the type of symmetric lists of *length* $n : \mathbb{N}$ with elements in $\mathbf{1}$.

Definition 5.32. We define a function

$$\mathfrak{k} : \mathsf{slist}(\mathbf{1}) \to \mathsf{del}_\bullet$$

using the elimination principle of slist. The arguments of $\mathsf{rec}_{\mathsf{slist}}$ are :

- a term $\mathsf{nil}' : \mathsf{del}_\bullet$, given by $\mathsf{nil}' :\equiv \underline{\mathsf{pt}_0} \equiv \langle 0, \mathsf{pt}_0 \rangle$;
- a term $\mathsf{cons}' : \mathbf{1} \to \mathsf{del}_\bullet \to \mathsf{del}_\bullet$, defined to be $\mathsf{cons}'_x :\equiv \langle \mathsf{s}, \mathsf{i} \rangle$ for every $x : \mathbf{1}$;
- for every $x, y : \mathbf{1}$ and $\langle n, a \rangle : \mathsf{del}_\bullet$, a path

  $$\mathsf{swap}'_{x,y,\langle n,a \rangle} : \mathsf{cons}'_x(\mathsf{cons}'_y\langle n, a \rangle) = \mathsf{cons}'_y(\mathsf{cons}'_x\langle n, a \rangle).$$

  Its type, unfolding the definition of cons', is judgmentally equal to

  $$\langle n+2, \mathsf{i}(\mathsf{i}(a)) \rangle = \langle n+2, \mathsf{i}(\mathsf{i}(a)) \rangle;$$

  the path can be then defined as $\mathsf{swap}'_{x,y,\langle n,a \rangle} :\equiv \langle \mathsf{refl}_{n+2}, \mathsf{tw}_a \rangle$;
- for every $x, y : \mathbf{1}$ and $\langle n, a \rangle : \mathsf{del}_\bullet$, a 2-path

  $$\mathsf{double}'_{x,y,\langle n,a \rangle} : \mathsf{swap}'_{x,y,\langle n,a \rangle} \cdot \mathsf{swap}'_{y,x,\langle n,a \rangle} = \mathsf{refl}_{\mathsf{cons}'_x(\mathsf{cons}'_y\langle n,a \rangle)}.$$

  Using Lemma 2.69 and do_a, we get:

  $$\begin{aligned} \mathsf{swap}'_{x,y,\langle n,a \rangle} \cdot \mathsf{swap}'_{y,x,\langle n,a \rangle} &\equiv \langle \mathsf{refl}_{n+2}, \mathsf{tw}_a \rangle \cdot \langle \mathsf{refl}_{n+2}, \mathsf{tw}_a \rangle \\ &= \langle \mathsf{refl}_{n+2}, \mathsf{tw}_a \cdot^{\mathsf{d}} \mathsf{tw}_a \rangle \\ &\equiv \langle \mathsf{refl}_{n+2}, \mathsf{tw}_a \cdot \mathsf{tw}_a \rangle \\ &= \langle \mathsf{refl}_{n+2}, \mathsf{refl}_{\mathsf{i}(\mathsf{i}(a))} \rangle \\ &\equiv \mathsf{refl}_{\langle n+2, \mathsf{i}(\mathsf{i}(a)) \rangle} \equiv \mathsf{refl}_{\mathsf{cons}'_x(\mathsf{cons}'_y\langle n,a \rangle)}; \end{aligned}$$

- for every $x, y, z : \mathbf{1}$ and $\langle n, a \rangle : \mathsf{del}_\bullet$, a 2-path

  $$\begin{aligned} \mathsf{triple}'_{x,y,z,\langle n,a \rangle} : \mathsf{swap}'_{x,y,\mathsf{cons}'_z\langle n,a \rangle} \cdot [\mathsf{cons}'_y](\mathsf{swap}'_{x,z,\langle n,a \rangle}) \cdot \mathsf{swap}'_{y,z,\mathsf{cons}'_x\langle n,a \rangle} \\ = [\mathsf{cons}'_x](\mathsf{swap}'_{y,z,\langle n,a \rangle}) \cdot \mathsf{swap}'_{x,z,\mathsf{cons}'_y\langle n,a \rangle} \cdot [\mathsf{cons}'_z](\mathsf{swap}'_{x,y,\langle n,a \rangle}), \end{aligned}$$

  similarly obtained by path algebra and by br_a;

- a proof that $\mathsf{del}_\bullet$ is a 1-type, which follows from the constructor T_{del} of $\mathsf{del}_{(-)}$, together with Remark 2.75.

Remark 5.33. It is tedious, but not hard, to prove that $\mathfrak{k}$ is a symmetric monoidal functor:

- the path $\mathfrak{k}_0 : \underline{\mathsf{pt}_0} = \mathfrak{k}(\mathsf{nil})$ is given by reflexivity;
- a family $\mathfrak{k}_2 : \prod(l_1, l_2 : \mathsf{slist}(\mathbf{1})).\, \mathfrak{k}(l_1) \oplus \mathfrak{k}(l_2) = \mathfrak{k}(l_1 +\!\!+ l_2)$ needs to be constructed by induction on l_1; on 0-constructors, we have

$$\mathfrak{k}_2(\mathsf{nil}, l_2) \equiv \mathsf{refl}_{\mathfrak{k}(l_2)}, \qquad \mathfrak{k}_2(x :: l_1, l_2) \equiv [\langle \mathsf{s}, \mathsf{i}\rangle](\mathfrak{k}_2(l_1, l_2))$$

  for every $x : \mathbf{1}$ and l_1, $l_2 : \mathsf{slist}(\mathbf{1})$, while the family of 2-paths relative to the 1-constructor swap derives from the fact that loops $\langle \mathsf{refl}, \mathsf{tw}_{(-)}\rangle$ commute with paths of the form $[\langle \mathsf{s}, \mathsf{i}\rangle]([\langle \mathsf{s}, \mathsf{i}\rangle](-))$, cf. Lemma 5.21(i);
- the families $\mathfrak{k}_\alpha$, $\mathfrak{k}_\lambda$, $\mathfrak{k}_\rho$ and $\mathfrak{k}_\tau$ are easily obtained via the computation rules of $\mathfrak{k}$ relative to the 0-constructors, as the defining terms of the symmetric monoidal structures of $\mathsf{slist}(\mathbf{1})$ and $\mathsf{del}_\bullet$ correspond in all aspects.

Definition 5.34. We define a function

$$\mathsf{j} : \mathsf{del}_\bullet \to \mathsf{slist}(\mathbf{1})$$

using the elimination principle of Σ-types (by which it is enough to provide a function $\prod(n : \mathbb{N}).\, \mathsf{del}_n \to \mathsf{slist}(\mathbf{1})$) and the one of $\mathsf{del}_{(-)}$. The arguments of $\mathsf{rec}_{\mathsf{del}}$ are the following:

- a term $\mathsf{pt}'_0 : \mathsf{slist}(\mathbf{1})$, defined to be $\mathsf{pt}'_0 :\equiv \mathsf{nil}$;
- a term $\mathsf{i}' : \mathbb{N} \to \mathsf{slist}(\mathbf{1}) \to \mathsf{slist}(\mathbf{1})$, given by $\mathsf{i}'_n :\equiv \mathsf{cons}(*)$ uniformly in $n : \mathbb{N}$;
- for every $n : \mathbb{N}$ and $l : \mathsf{slist}(\mathbf{1})$, a path

$$\mathsf{tw}'_n(l) : \mathsf{i}'_{n+1}(\mathsf{i}'_n(l)) = \mathsf{i}'_{n+1}(\mathsf{i}'_n(l)),$$

  the type of which, unfolding the definition of i', is judgmentally equal to

$$* :: * :: l = * :: * :: l;$$

  in view of the fact that we want to construct a half-adjoint inverse to $\mathfrak{k}$, we define this to be $\mathsf{tw}'_n(l) :\equiv \mathsf{swap}_{*,*,l}$;
- for every $n : \mathbb{N}$ and $l : \mathsf{slist}(\mathbf{1})$, a 2-path

$$\mathsf{do}'_n(l) : \mathsf{tw}'_n(l) \cdot \mathsf{tw}'_n(l) = \mathsf{refl}_{\mathsf{i}'_{n+1}(\mathsf{i}'_n(l))},$$

  given by $\mathsf{do}'_n(l) :\equiv \mathsf{double}_{*,*,l}$;

- for every $n : \mathbb{N}$ and $l : \mathsf{slist}(\mathbf{1})$, a 2-path

$$\begin{aligned}\mathsf{br}'_n(l) : \mathsf{tw}'_{n+1}(\mathsf{i}'_n(l)) \cdot [\mathsf{i}'_{n+2}](\mathsf{tw}'_n(l)) \cdot \mathsf{tw}'_{n+1}(\mathsf{i}'_n(l)) \\ = [\mathsf{i}'_{n+2}](\mathsf{tw}'_n(l)) \cdot \mathsf{tw}'_{n+1}(\mathsf{i}'_n(l)) \cdot [\mathsf{i}'_{n+2}](\mathsf{tw}'_n(l)),\end{aligned}$$

  given by $\mathsf{br}'_n(l) :\equiv \mathsf{triple}_{*,*,*,l}$;

- a proof that $\mathsf{slist}(\mathbf{1})$ is a 1-type, which is in its definition.

Theorem 5.35. *The functions $\mathfrak{k}$ and j presented in Definitions 5.32 and 5.34 are half-adjoint in an equivalence, which can be promoted to a symmetric monoidal equivalence*

$$\mathsf{slist}(\mathbf{1}) \simeq \mathsf{del}_\bullet.$$

Proof. Homotopies $\mathsf{j} \circ \mathfrak{k} \sim \mathsf{id}_{\mathsf{slist}(\mathbf{1})}$ and $\mathfrak{k} \circ \mathsf{j} \sim \mathsf{id}_{\mathsf{del}_\bullet}$ are obtained because the functions j and $\mathfrak{k}$ inductively pair the constructors of slist with those of $\mathsf{del}_{(-)}$. The first homotopy is provided as follows: a function $\Pi\,(l : \mathsf{slist}(\mathbf{1}))\,.\,\mathsf{j}(\mathfrak{k}(l)) = l$ is produced by the elimination principle of slist, following the scheme for elimination in a family of paths in a groupoid. The required arguments of $\mathsf{ind}_{\mathsf{slist}}$ are:

- a path $\mathsf{nil}' : \mathsf{j}(\mathfrak{k}(\mathsf{nil})) = \mathsf{nil}$, given by $\mathsf{refl}_{\mathsf{nil}}$, as the left-hand side of the identity computes to nil;
- given $x : \mathbf{1}$, $l : \mathsf{slist}(\mathbf{1})$ and assuming the inductive hypothesis $h : \mathsf{j}(\mathfrak{k}(l)) = l$, a path $\mathsf{cons}'_x(l,h) : \mathsf{j}(\mathfrak{k}(x :: l)) = x :: l$. By the elimination principle of $\mathbf{1}$, it is enough to provide a path $\mathsf{cons}'_*(l,h) : \mathsf{j}(\mathfrak{k}(* :: l)) = * :: l$; as the left-hand side of the identity computes to $* :: \mathsf{j}(\mathfrak{k}(l))$, the claim is proved by $[* :: -](h)$;
- for $x, y : \mathbf{1}$, $l : \mathsf{slist}(\mathbf{1})$ and assuming the inductive hypothesis $h : \mathsf{j}(\mathfrak{k}(l)) = l$, the 2-path $\mathsf{swap}'_{x,y}(l,h)$ shown in Fig. 5.4a; by the elimination principle of $\mathbf{1}$, it is enough to give $\mathsf{swap}'_{*,*}(l,h)$, which is shown in Fig. 5.4b to be obtained by the computation rules of j and $\mathfrak{k}$ and by naturality of the constructor swap.

A homotopy $\mathfrak{k} \circ \mathsf{j} \sim \mathsf{id}_{\mathsf{del}_\bullet}$ is obtained in a completely similar fashion: the composite function $\mathfrak{k} \circ \mathsf{j}$:

- leaves $\langle 0, \mathsf{pt}_0 \rangle$ fixed (judgmentally);
- sends any term of the form $\langle n+1, \mathsf{i}(a) \rangle$ to $\langle \mathsf{s}, \mathsf{i} \rangle (\mathfrak{k}(\mathsf{j}\langle n, a \rangle))$ which is by induction equal to $\langle \mathsf{s}, \mathsf{i} \rangle \langle n, a \rangle \equiv \langle n+1, \mathsf{i}(a) \rangle$;
- maps functorially any path of the form $\langle \mathsf{refl}_{n+2}, \mathsf{tw}_a \rangle$ to a pathover corresponding to a 2-path in $\mathsf{del}_\bullet$ given by the computation rules of $\mathfrak{k}$ and j on 1-constructors and naturality of tw.

$$\begin{array}{ccc} \mathsf{j}(\mathfrak{k}(x :: y :: l)) & \xrightarrow{\mathsf{cons}'_x(y :: l, \mathsf{cons}'_y(l,h))} & x :: y :: l \\ {\scriptstyle [\mathsf{j}]([\mathfrak{k}](\mathsf{swap}_{x,y,l}))}\Big\downarrow & & \Big\downarrow{\scriptstyle \mathsf{swap}_{x,y,l}} \\ \mathsf{j}(\mathfrak{k}(y :: x :: l)) & \xrightarrow[\mathsf{cons}'_y(x :: l, \mathsf{cons}'_x(l,h))]{} & y :: x :: l \end{array}$$

(a) The type of the 2-path $\mathsf{swap}'_{x,y}(l,h)$, which will be obtained by induction on $x, y : \mathbf{1}$.

$$\begin{array}{ccc} * :: * :: \mathsf{j}(\mathfrak{k}(l)) & \xrightarrow{[* :: -]([* :: -](h))} & * :: * :: l \\ {\scriptstyle [\mathsf{j}]([\mathfrak{k}](\mathsf{swap}_{*,*,l}))}\Big\downarrow \; (1) & \Big\Downarrow{\scriptstyle \mathsf{swap}_{*,*,\mathsf{j}(\mathfrak{k}(l))}} \quad (2) & \Big\downarrow{\scriptstyle \mathsf{swap}_{*,*,l}} \\ * :: * :: \mathsf{j}(\mathfrak{k}(l)) & \xrightarrow[[* :: -]([* :: -](h))]{} & * :: * :: l \end{array}$$

(b) The 2-path $\mathsf{swap}_{*,*}(l,h)$, after unfolding the definition of cons' and computing j and $\mathfrak{k}$ in the terms on the left side. The 2-path (1) is obtained by the computation rules of j and $\mathfrak{k}$; (2) is given by naturality of swap.

Figure 5.4: The 2-path $\mathsf{swap}'_{x,y}(l,h)$ in the definition of the homotopy $\mathsf{j} \circ \mathfrak{k} \sim \mathsf{id}_{\mathsf{slist}(\mathbf{1})}$.

As $\mathsf{del}_\bullet$ is a Σ-type, the path algebra employed to prove the last point is quite cumbersome; a full proof is present in the Coq formalization. The symmetric monoidal equivalence is achieved by virtue of Remark 4.9. □

5.4 A Degreewise Equivalence $\mathsf{del}_\bullet \simeq \mathcal{BS}_\bullet$

Our next goal, which is only partially achieved, is to construct a symmetric monoidal equivalence $\mathsf{del}_\bullet \simeq \mathcal{BS}_\bullet$. This can be obtained from a family of equivalences

$$\mathfrak{f} : \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \simeq \mathcal{BS}_n. \tag{5.36}$$

Providing such a family of equivalences would imply that the combinatorial structure described in Section 5.2 characterizes the subuniverse, and hence that all paths between finite types are *generated* by instances of ω, via univalence.

It is immediate to see why such a result would be desirable. A function out of $\mathcal{BS}_{(-)}$ cannot, in principle, be produced in an inductive way, as the universe $\mathcal{U}$, which is the base of the Σ-type defining $\mathcal{BS}_n$, is not an inductive type and does not possess an elimination principle. In contrast, Definition 5.18 specifies the requirements to obtain indexed functions out of $\mathsf{del}_{(-)}$. An equivalence $\mathsf{del}_\bullet \simeq \mathcal{BS}_\bullet$ would

bridge this gap, effectively providing an elimination principle for finite types. At the same time, the difficulty in constructing functions out of $\mathcal{BS}_{(-)}$ will constitute the main challenge to overcome in order to construct a family of equivalences as in (5.36).

We begin by specializing the non-dependent elimination principle $\mathsf{rec}_{\mathsf{del}}$ of $\mathsf{del}_{(-)}$ to the case in which we eliminate into a Σ-type whose family is a family of (-1)-types; we will call the ensuing function $\mathsf{rec}^{\Sigma}_{\mathsf{del}}$.

Lemma 5.37. *Let $A : \mathcal{U}$ and $Q : \mathbb{N} \to A \to \mathcal{U}$, and assume given the terms*

$$T'_{\mathsf{del}} : \Pi\,(n : \mathbb{N})\,.\ \mathsf{IsHGpd}(\Sigma\,(a : A)\,.\,Q_n(a)),$$
$$r : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : A)\,.\ \mathsf{IsHProp}(Q_n(a)).$$

In order to construct a function $\mathsf{rec}^{\Sigma}_{\mathsf{del}} : \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \to \Sigma\,(a : A)\,.\,Q_n(a)$, it suffices to provide the following terms:[3]

$$\mathsf{pt}^{\mathsf{b}}_0 : A, \qquad \mathsf{pt}^{\mathsf{f}}_0 : Q_0(\mathsf{pt}^{\mathsf{b}}_0),$$
$$\mathsf{i}^{\mathsf{b}} : \mathbb{N} \to A \to A, \qquad \mathsf{i}^{\mathsf{f}} : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : A)\,.\,Q_n(a) \to Q_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a)),$$
$$\mathsf{tw}^{\mathsf{b}} : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : A)\,.\,\mathsf{i}^{\mathsf{b}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a)) = \mathsf{i}^{\mathsf{b}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a)),$$
$$\mathsf{do}^{\mathsf{b}} : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : A)\,.\,\mathsf{tw}^{\mathsf{b}}_n(a) \cdot \mathsf{tw}^{\mathsf{b}}_n(a) = \mathsf{refl}_{\mathsf{i}^{\mathsf{b}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a))},$$
$$\mathsf{br}^{\mathsf{b}} : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : A)\,.\,\mathsf{tw}^{\mathsf{b}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a)) \cdot [\mathsf{i}^{\mathsf{b}}_{n+2}](\mathsf{tw}^{\mathsf{b}}_n(a)) \cdot \mathsf{tw}^{\mathsf{b}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a))$$
$$= [\mathsf{i}^{\mathsf{b}}_{n+2}](\mathsf{tw}^{\mathsf{b}}_n(a)) \cdot \mathsf{tw}^{\mathsf{b}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a)) \cdot [\mathsf{i}^{\mathsf{b}}_{n+2}](\mathsf{tw}^{\mathsf{b}}_n(a)).$$

Proof. By the non-dependent elimination principle $\mathsf{rec}_{\mathsf{del}}$ of $\mathsf{del}_{(-)}$, the function $\mathsf{rec}^{\Sigma}_{\mathsf{del}}$ can be constructed by providing:

- a term $\mathsf{pt}'_0 : \Sigma\,(a : A)\,.\,Q_0(a)$, which can be given by $\langle \mathsf{pt}^{\mathsf{b}}_0, \mathsf{pt}^{\mathsf{f}}_0 \rangle$;
- for every $n : \mathbb{N}$, a function $\mathsf{i}'_n : (\Sigma\,(a : A)\,.\,Q_n(a)) \to (\Sigma\,(a : A)\,.\,Q_{n+1}(a))$; this is given by $\langle \mathsf{i}^{\mathsf{b}}_n, \mathsf{i}^{\mathsf{f}}_n \rangle$;
- for every $n : \mathbb{N}$, $a : A$ and $q : Q_n(a)$, a path in $\Sigma\,(a : A)\,.\,Q_{n+2}(a)$

  $$\mathsf{tw}'_n\langle a, q\rangle : \mathsf{i}'_{n+1}(\mathsf{i}'_n\langle a, q\rangle) = \mathsf{i}'_{n+1}(\mathsf{i}'_n\langle a, q\rangle).$$

  By our definition of i', the identity type is judgmentally equal to

  $$\langle \mathsf{i}^{\mathsf{b}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a)),\ \mathsf{i}^{\mathsf{f}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a), \mathsf{i}^{\mathsf{f}}_n(a, q))\rangle = \langle \mathsf{i}^{\mathsf{b}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a)),\ \mathsf{i}^{\mathsf{f}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a), \mathsf{i}^{\mathsf{f}}_n(a, q))\rangle;$$

  as r guarantees that $Q_{n+2}(a)$ is a (-1)-type, the given path $\mathsf{tw}^{\mathsf{b}}_n(a)$ in the base of the Σ-type suffices (Remark 2.78);

[3] The notation $-^{\mathsf{b}}$ and $-^{\mathsf{f}}$ refers to *b*ase and *f*ibers of a Σ-type.

- for every $n : \mathbb{N}$, $a : A$ and $q : Q_n(a)$, a 2-path in $\Sigma\,(a : A)\,.\,Q_{n+2}(a)$

$$\mathsf{do}'_n\langle a, q\rangle : \mathsf{tw}'_n\langle a, q\rangle \cdot \mathsf{tw}'_n\langle a, q\rangle = \mathsf{refl}_{\mathsf{i}'_{n+1}(\mathsf{i}'_n\langle a,q\rangle)}.$$

  The type of $\mathsf{do}'_n\langle a, q\rangle$, after unfolding the definition of tw'_n, is:

$$\langle \mathsf{tw}^{\mathsf{b}}_n(a), \ldots\rangle \cdot \langle \mathsf{tw}^{\mathsf{b}}_n(a), \ldots\rangle = \langle \mathsf{refl}_{\mathsf{i}^{\mathsf{b}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a))}, \ldots\rangle.$$

  By Lemma 2.69, it is enough to provide a 2-path

$$\langle \mathsf{tw}^{\mathsf{b}}_n(a) \cdot \mathsf{tw}^{\mathsf{b}}_n(a), \ldots\rangle = \langle \mathsf{refl}_{\mathsf{i}^{\mathsf{b}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a))}, \ldots\rangle;$$

  using again Remark 2.78, by virtue of r, $\mathsf{do}^{\mathsf{b}}_n(A)$ suffices;

- for every $n : \mathbb{N}$, $a : A$ and $q : Q_n(a)$, a 2-path in $\Sigma\,(a : A)\,.\,Q_{n+3}(a)$

$$\begin{aligned}\mathsf{br}'_n\langle a, q\rangle : \mathsf{tw}'_{n+1}(\mathsf{i}'_n\langle a, q\rangle) \cdot [\mathsf{i}'_{n+2}](\mathsf{tw}'_n\langle a, q\rangle) \cdot \mathsf{tw}'_{n+1}(\mathsf{i}'_n\langle a, q\rangle)\\ = [\mathsf{i}'_{n+2}](\mathsf{tw}'_n\langle a, q\rangle) \cdot \mathsf{tw}'_{n+1}(\mathsf{i}'_n\langle a, q\rangle) \cdot [\mathsf{i}'_{n+2}](\mathsf{tw}'_n\langle a, q\rangle).\end{aligned}$$

  Unfolding the definition of i' and of tw', and applying Lemma 2.69, this entails finding a term in

$$\begin{aligned}&\langle \mathsf{tw}^{\mathsf{b}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a)) \cdot [\mathsf{i}^{\mathsf{b}}_{n+2}](\mathsf{tw}^{\mathsf{b}}_n(a)) \cdot \mathsf{tw}^{\mathsf{b}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a)), \ldots\rangle\\ &= \langle [\mathsf{i}^{\mathsf{b}}_{n+2}](\mathsf{tw}^{\mathsf{b}}_n(a)) \cdot \mathsf{tw}^{\mathsf{b}}_{n+1}(\mathsf{i}^{\mathsf{b}}_n(a)) \cdot [\mathsf{i}^{\mathsf{b}}_{n+2}](\mathsf{tw}^{\mathsf{b}}_n(a)), \ldots\rangle,\end{aligned}$$

  which is given by $\mathsf{br}^{\mathsf{b}}_n(a)$, using Remark 2.78;

- the term T'_{del} itself, witnessing the truncation level of the Σ-type. □

Remark 5.38. The function $\mathsf{rec}^{\Sigma}_{\mathsf{del}}$ in Lemma 5.37 inherits the computation properties of the function $\mathsf{rec}_{\mathsf{del}}$ by which it is defined, i.e., we have:

$$\mathsf{rec}^{\Sigma}_{\mathsf{del}}(0, \mathsf{pt}_0) \equiv \langle \mathsf{pt}^{\mathsf{b}}_0, \mathsf{pt}^{\mathsf{f}}_0\rangle, \qquad \mathsf{rec}^{\Sigma}_{\mathsf{del}}(n+1, \mathsf{i}(a)) \equiv \langle \mathsf{i}^{\mathsf{b}}, \mathsf{i}^{\mathsf{f}}\rangle(\mathsf{rec}^{\Sigma}_{\mathsf{del}}(n, a)),$$

and so on.

There is no hindrance to the proof of the lemma above when A is a universe of types. If this is the case, we are able to define functions $\Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \to \mathcal{U}$, provided that they factor through an $\mathbb{N}$-indexed family of subuniverses, all of which are 1-types:

$$\mathsf{del}_{(-)} \xrightarrow{\mathsf{rec}^{\Sigma}_{\mathsf{del}}} \Sigma\,(A : \mathcal{U})\,.\,Q_{(-)}(A) \xrightarrow{\mathsf{pr}_1} \mathcal{U}.$$

We will make use of this observation and employ the subuniverse of finite types as factoring term.

Definition 5.39. We construct a function

$$\mathfrak{f} : \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \to \mathcal{BS}_n.$$

In order to do so, we will use $\mathsf{rec}^{\Sigma}_{\mathsf{del}}$ defined in Lemma 5.37, applied to $A :\equiv \mathcal{U}$ and $Q :\equiv (n \mapsto (A \mapsto \|A \simeq [n]\|))$. We then need to provide the following arguments:

- a proof that $\mathcal{BS}_n$ is a 1-type for every n; this is shown in Corollary 5.7;
- a proof that $\|A \simeq [n]\|$ is a (-1)-type for every n and A; this is immediate;
- a term $\mathsf{pt}^{\mathsf{b}}_0 : \mathcal{U}$, which we choose to be $\mathbf{0} \equiv [0]$;
- a term $\mathsf{pt}^{\mathsf{f}}_0 : \|[0] \simeq [0]\|$, such is $|\mathsf{id}_{[0]}|$;
- a function $\mathsf{i}^{\mathsf{b}} : \mathbb{N} \to \mathcal{U} \to \mathcal{U}$; this will be constructed uniformly in $\mathbb{N}$, using Definition 2.42, as $\mathsf{i}^{\mathsf{b}}_n :\equiv \mathsf{add} \equiv (X \mapsto X + \mathbf{1})$;
- for every $n : \mathbb{N}$ and $A : \mathcal{U}$, a function $\mathsf{i}^{\mathsf{f}}_n(A) : \|A \simeq [n]\| \to \|(A + \mathbf{1}) \simeq [n+1]\|$ where, we recall, $[n+1] \equiv [n] + \mathbf{1}$. By the elimination principle of the truncation, the function

  $$(e \mapsto |\mathsf{incr}(e)|) : (A \simeq [n]) \to \|(A + \mathbf{1}) \simeq ([n] + \mathbf{1})\|\,,$$

  with incr as in Definition 2.100, suffices;
- for every $n : \mathbb{N}$ and $A : \mathcal{U}$, a path $\mathsf{tw}^{\mathsf{b}}_n(A) : A + \mathbf{1} + \mathbf{1} = A + \mathbf{1} + \mathbf{1}$. This can be given by $\mathsf{tw}^{\mathsf{b}}_n(A) :\equiv \gamma_A$, defined in Remark 5.16;
- for every $n : \mathbb{N}$ and $A : \mathcal{U}$, a 2-path $\mathsf{do}^{\mathsf{b}}_n : \gamma_A \cdot \gamma_A = \mathsf{refl}_{A+\mathbf{1}+\mathbf{1}}$. We have a chain of identities:

  $$\gamma_A \cdot \gamma_A \equiv \mathsf{ua}(\omega_A) \cdot \mathsf{ua}(\omega_A) = \mathsf{ua}(\omega_A \circ \omega_A) = \mathsf{ua}(\mathsf{id}_{A+\mathbf{1}+\mathbf{1}}) = \mathsf{refl}_{A+\mathbf{1}+\mathbf{1}}$$

  using Lemma 2.113 and the fact that ω_A is an involution;
- for every $n : \mathbb{N}$ and $A : \mathcal{U}$, a 2-path

  $$\mathsf{br}^{\mathsf{b}}_n : \gamma_{A+\mathbf{1}} \cdot [\mathsf{add}](\gamma_A) \cdot \gamma_{A+\mathbf{1}} = [\mathsf{add}](\gamma_A) \cdot \gamma_{A+\mathbf{1}} \cdot [\mathsf{add}](\gamma_A).$$

  Using Lemmata 2.113 and 2.127, this reduces to finding a 2-path

  $$\mathsf{ua}(\omega_{A+\mathbf{1}} \circ \mathsf{incr}(\omega_A) \circ \omega_{A+\mathbf{1}}) = \mathsf{ua}(\mathsf{incr}(\omega_A) \circ \omega_{A+\mathbf{1}} \circ \mathsf{incr}(\omega_A)),$$

  which is obtained from (5.15) by application of ua.

The computational content of $\mathfrak{f}$ is, in this case, more interesting after taking the projection to $\mathcal{U}$. Naming

$$\mathfrak{f}^{\mathsf{b}}_n :\equiv \mathsf{pr}_1 \circ \mathfrak{f}_n : \mathsf{del}_n \to \mathcal{U},$$

we have, because of the computation rules of $\mathfrak{f}$:

$$\mathfrak{f}^{\mathsf{b}}_0(\mathsf{pt}_0) \equiv \mathbf{0}, \tag{5.40}$$

$$\mathfrak{f}^{\mathsf{b}}_{n+1}(\mathsf{i}(a)) \equiv \mathfrak{f}^{\mathsf{b}}_n(a) + \mathbf{1} \quad \text{for every } n : \mathbb{N},\ a : \mathsf{del}_n, \tag{5.41}$$

$$\text{and a path } [\mathfrak{f}^{\mathsf{b}}_{n+2}](\mathsf{tw}_a) = \gamma_{\mathfrak{f}^{\mathsf{b}}_n(a)} \quad \text{for every } n : \mathbb{N},\ a : \mathsf{del}_n. \tag{5.42}$$

We will not be need the computation rules relative to the constructors do and br of $\mathsf{del}_{(-)}$, since $\mathfrak{f}$ eliminates into a family of 1-types.

Lemma 5.43. *Let $n : \mathbb{N}$; $a, b : \mathsf{del}_n$ and $p : a = b$. There is a 2-path*

$$[\mathfrak{f}^{\mathsf{b}}_{n+1}]([\mathsf{i}](p)) =_{(\mathfrak{f}^{\mathsf{b}}_n(a)+\mathbf{1}=\mathfrak{f}^{\mathsf{b}}_n(b)+\mathbf{1})} [\mathsf{add}]([\mathfrak{f}^{\mathsf{b}}_n](p)), \tag{5.44}$$

as shown in Fig. 5.5; the expression above is well-typed because of (5.41).

Proof. By induction on p. □

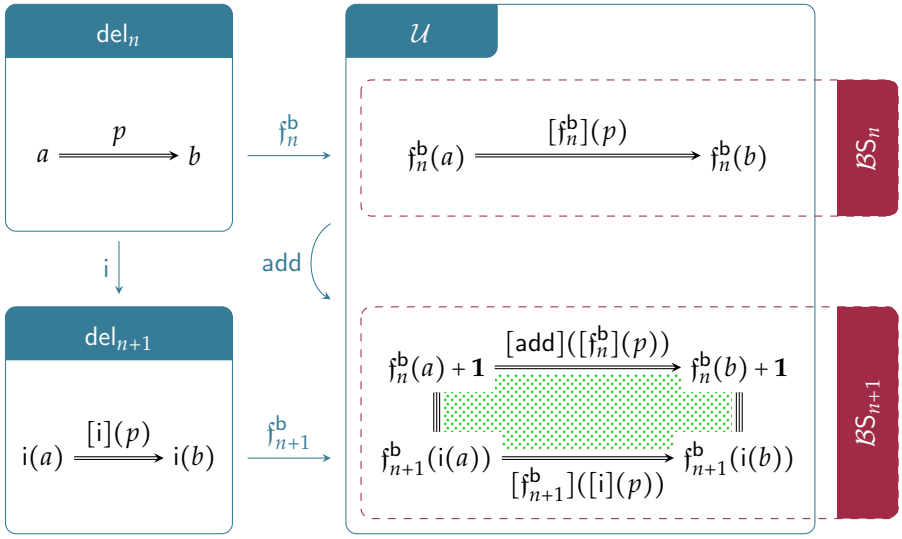


Figure 5.5: The 2-path in (5.44), in a scheme showing the relationship between i and add.

Remark 5.45. A function $\mathsf{r}^{\mathsf{b}} : \Pi\,(n : \mathbb{N})\,.\,\mathfrak{f}^{\mathsf{b}}_n(\mathsf{pt}_n) = [n]$ is defined by the elimination principle of $\mathbb{N}$, with the assignments $\mathsf{r}^{\mathsf{b}}_0 :\equiv \mathsf{refl}_{[0]}$ and $\mathsf{r}^{\mathsf{b}}_{n+1} :\equiv [\mathsf{add}](\mathsf{r}^{\mathsf{b}}_n)$, which are

well-defined because of the computation rules in (5.40) and (5.41), respectively. The family r^{b} of paths is enough to define a family $\mathsf{r} : \Pi\,(n : \mathbb{N})\,.\,\mathfrak{f}_n(\mathsf{pt}_n) = \langle [n], |\mathsf{id}_{[n]}| \rangle$, by virtue of Remark 2.78. This implies, in particular, that the features of canonical finite types described in Section 5.1 are also enjoyed by the type $\mathfrak{f}^{\mathsf{b}}_n(\mathsf{pt}_n) : \mathcal{U}$. We will use this type as the preferred finite type of cardinality n, rather than the canonical finite type $[n] : \mathcal{U}$; to unburden the syntax in this section and emphasize the relation between $\mathfrak{f}^{\mathsf{b}}_n(\mathsf{pt}_n)$ and $[n]$, we will use the notation

$$[\![n]\!] :\equiv \mathfrak{f}^{\mathsf{b}}_n(\mathsf{pt}_n),$$

for every $n : \mathbb{N}$. Note that, by (5.41), $[\![n+1]\!] \equiv [\![n]\!] + \mathbf{1}$ for every $n : \mathbb{N}$, and that $[\![-]\!]$ *computes* at numerals: we have, e.g., $[\![0]\!] \equiv [0] \equiv \mathbf{0}$ by (5.40), $[\![1]\!] \equiv [1]$, $[\![2]\!] \equiv [2]$, and so on.

The function $\mathfrak{f}^{\mathsf{b}} : \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \to \mathcal{U}$ allows us to prove that $\mathsf{tw}_{(-)}$ produces nontrivial loops.

Lemma 5.46. *For any $n : \mathbb{N}$ and $a : \mathsf{del}_n$, there is no 2-path $p : \mathsf{tw}_a = \mathsf{refl}_{\mathsf{i}(\mathsf{i}(a))}$ in del_{n+2}, i.e., there is a function $(\mathsf{tw}_a = \mathsf{refl}_{\mathsf{i}(\mathsf{i}(a))}) \to \mathbf{0}$.*

Proof. The argument is analogous to the one presented in [Uni13, Lemma 6.4.1] to prove that there is no 2-path $\mathsf{loop} =_{\mathsf{S}^1} \mathsf{refl}_{\mathsf{base}}$. Given $n : \mathbb{N}$ and $a : \mathsf{del}_n$, if there were a 2-path $p : \mathsf{tw}_a = \mathsf{refl}_{\mathsf{i}(\mathsf{i}(a))}$, then we would have, using the computation rules (5.41) and (5.42) of $\mathfrak{f}^{\mathsf{b}}$ and Lemma 2.113, a 2-path

$$\begin{aligned}\mathsf{ua}(\omega_{\mathfrak{f}^{\mathsf{b}}_n(a)}) \equiv \gamma_{\mathfrak{f}^{\mathsf{b}}_n(a)} = [\mathfrak{f}^{\mathsf{b}}_{n+2}](\mathsf{tw}_a) &= [\mathfrak{f}^{\mathsf{b}}_{n+2}](\mathsf{refl}_{\mathsf{i}(\mathsf{i}(a))}) \\ &= \mathsf{refl}_{\mathfrak{f}^{\mathsf{b}}_{n+2}(\mathsf{i}(\mathsf{i}(a)))} = \mathsf{refl}_{\mathfrak{f}^{\mathsf{b}}_n(a)+\mathbf{1}+\mathbf{1}} = \mathsf{ua}(\mathsf{id}_{\mathfrak{f}^{\mathsf{b}}_n(a)+\mathbf{1}+\mathbf{1}}).\end{aligned}$$

As ua is an equivalence, this would imply a path $\omega_{\mathfrak{f}^{\mathsf{b}}_n(a)} = \mathsf{id}_{\mathfrak{f}^{\mathsf{b}}_n(a)+\mathbf{1}+\mathbf{1}}$, and in particular a path $\mathsf{inl}(\mathsf{inr}(*)) \equiv \omega_{\mathfrak{f}^{\mathsf{b}}_n(a)}(\mathsf{inr}(*)) = \mathsf{id}_{\mathfrak{f}^{\mathsf{b}}_n(a)+\mathbf{1}+\mathbf{1}}(\mathsf{inr}(*)) \equiv \mathsf{inr}(*)$. Lemma 2.70 provides the contradiction. □

The lemma above has the following consequence.

Corollary 5.47. *For no $n : \mathbb{N}$ is del_{n+2} contractible.*

Proof. Let $n : \mathbb{N}$; by Lemma 5.46, there is no 2-path $\mathsf{tw}_{\mathsf{pt}_n} = \mathsf{refl}_{\mathsf{pt}_{n+2}}$, so del_{n+2} is not a 0-type, and hence not a (-2)-type. □

A desired feature of the family $\mathfrak{f} : \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \to \mathcal{BS}_n$ of functions defined above is that it respect the symmetric monoidal structures of $\mathsf{del}_\bullet$ and $\mathcal{BS}_\bullet$ given in Lemma 5.30 and Lemma 5.11.

Definition 5.48. The function $\mathfrak{f}_\bullet : \mathsf{del}_\bullet \to \mathcal{BS}_\bullet$ is defined by $\mathfrak{f}_\bullet :\equiv \langle \mathsf{id}_{\mathbb{N}}, \mathfrak{f}\rangle$, that is, for every $n : \mathbb{N}$ and $a : \mathsf{del}_n$, by

$$\mathfrak{f}_\bullet\langle n, a\rangle :\equiv \langle n, \mathfrak{f}_n(a)\rangle \equiv \langle n, \mathfrak{f}^{\mathsf{b}}_n(a), \ldots\rangle.$$

The function $\mathfrak{f}^{\mathsf{b}}_\bullet : \mathsf{del}_\bullet \to \mathcal{U}$ is defined by $\mathfrak{f}^{\mathsf{b}}_\bullet\langle n, a\rangle :\equiv \mathfrak{f}^{\mathsf{b}}_n(a)$.

Remark 5.49. For every $x : \mathsf{del}_\bullet$, we have

$$\mathfrak{f}_\bullet(\langle \mathsf{s}, \mathsf{i}\rangle(x)) = \langle \mathsf{s}, \mathsf{add}, \ldots\rangle(\mathfrak{f}_\bullet(x)), \tag{5.50}$$

where the omitted term is i^{f} from Definition 5.39. Indeed, using the elimination principle for Σ-types, we can verify, for every $n : \mathbb{N}$ and $a : \mathsf{del}_n$,

$$\begin{aligned}\mathfrak{f}_\bullet(\langle \mathsf{s}, \mathsf{i}\rangle\langle n, a\rangle) &\equiv \mathfrak{f}_\bullet\langle n+1, \mathsf{i}(a)\rangle \equiv \langle n+1, \mathfrak{f}_{n+1}(\mathsf{i}(a))\rangle \\ &\equiv \langle n+1, \mathfrak{f}^{\mathsf{b}}_n(a) + \mathbf{1}, \mathsf{i}^{\mathsf{f}}_n(\mathfrak{f}^{\mathsf{b}}_n(a))\rangle \\ &\equiv \langle \mathsf{s}, \mathsf{add}, \mathsf{i}^{\mathsf{f}}\rangle\langle n, \mathfrak{f}^{\mathsf{b}}_n(a), \ldots\rangle \equiv \langle \mathsf{s}, \mathsf{add}, \mathsf{i}^{\mathsf{f}}\rangle(\mathfrak{f}_\bullet\langle n, a\rangle).\end{aligned}$$

Similarly, on types, we have an identity

$$\mathfrak{f}^{\mathsf{b}}_\bullet(\langle \mathsf{s}, \mathsf{i}\rangle(x)) = \mathfrak{f}^{\mathsf{b}}_\bullet(x) + \mathbf{1},$$

as we can verify, for every $n : \mathbb{N}$ and $a : \mathsf{del}_n$,

$$\mathfrak{f}^{\mathsf{b}}_\bullet(\langle \mathsf{s}, \mathsf{i}\rangle\langle n, a\rangle) \equiv \mathfrak{f}^{\mathsf{b}}_\bullet\langle n+1, \mathsf{i}(a)\rangle \equiv \mathfrak{f}^{\mathsf{b}}_{n+1}(\mathsf{i}(a)) \equiv \mathfrak{f}^{\mathsf{b}}_n(a) + \mathbf{1} \equiv \mathfrak{f}^{\mathsf{b}}_\bullet\langle n, a\rangle + \mathbf{1}.$$

Theorem 5.51. *The function $\mathfrak{f}_\bullet : \mathsf{del}_\bullet \to \mathcal{BS}_\bullet$ is a symmetric monoidal functor.*

Proof. A path $(\mathfrak{f}_\bullet)_0 : \underline{0} = \mathfrak{f}_\bullet(\underline{\mathsf{pt}_0})$ is given by the reflexivity path, as both sides of the identity type compute to $\langle 0, \mathfrak{f}_0(\mathsf{pt}_0)\rangle$. For a family of paths

$$(\mathfrak{f}_\bullet)_2 : \Pi\,(x, y : \mathsf{del}_\bullet)\,.\,\mathfrak{f}_\bullet(x) \boxplus \mathfrak{f}_\bullet(y) = \mathfrak{f}_\bullet(x \oplus y),$$

we use the elimination principle of Σ-types and search for a family

$$\Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : \mathsf{del}_n)\,.\,\Pi\,(m : \mathbb{N})\,.\,\Pi\,(b : \mathsf{del}_m)\,.\,\langle n+m,\ \mathfrak{f}^{\mathsf{b}}_n(a) + \mathfrak{f}^{\mathsf{b}}_m(b), \ldots\rangle = \mathfrak{f}_\bullet(\langle n, a\rangle \oplus \langle m, b\rangle).$$

By the elimination principle of $\mathsf{del}_{(-)}$:

- for every $m : \mathbb{N}$ and $b : \mathsf{del}_m$, we have:

$$\begin{aligned}\langle 0+m,\ \mathbf{0} + \mathfrak{f}^{\mathsf{b}}_m(b), \ldots\rangle &\equiv \langle m,\ \mathbf{0} + \mathfrak{f}^{\mathsf{b}}_m(b), \ldots\rangle \\ &= \langle m,\ \mathfrak{f}^{\mathsf{b}}_m(b), \ldots\rangle && \text{by } \langle \mathsf{refl},\ \lambda_{\mathcal{U}}(\mathfrak{f}^{\mathsf{b}}_m(b)), \ldots\rangle \\ &\equiv \mathfrak{f}_\bullet\langle m, b\rangle \\ &\equiv \mathfrak{f}_\bullet(\langle 0, \mathsf{pt}_0\rangle \oplus \langle m, b\rangle) && \text{by (5.29)},\end{aligned}$$

  where $\lambda_{\mathcal{U}}$ is the left unit law for the coproduct of types, discussed in Section 5.1;

- given $n : \mathbb{N}$, $a : \mathsf{del}_n$ and a term

$$h : \Pi\,(m : \mathbb{N})\,.\,\Pi\,(b : \mathsf{del}_m)\,.\,\langle n+m, \mathfrak{f}^{\mathsf{b}}_n(a) + \mathfrak{f}^{\mathsf{b}}_m(b), \ldots\rangle = \mathfrak{f}_\bullet(\langle n, a\rangle \oplus \langle m, b\rangle), \quad (5.52)$$

  we have, for every $m : \mathbb{N}$ and $b : \mathsf{del}_m$,

$$\begin{aligned}
&\langle n+m+1,\ \mathfrak{f}^{\mathsf{b}}_n(a) + \mathbf{1} + \mathfrak{f}^{\mathsf{b}}_m(b), \ldots\rangle \\
=\ &\langle n+m+1,\ \mathfrak{f}^{\mathsf{b}}_n(a) + (\mathbf{1} + \mathfrak{f}^{\mathsf{b}}_m(b)), \ldots\rangle && \text{by } \langle \mathsf{refl},\ \alpha_{\mathcal{U}}(\mathfrak{f}^{\mathsf{b}}_n(a), \mathbf{1}, \mathfrak{f}^{\mathsf{b}}_m(b)), \ldots\rangle \\
=\ &\langle n+m+1,\ \mathfrak{f}^{\mathsf{b}}_n(a) + (\mathfrak{f}^{\mathsf{b}}_m(b) + \mathbf{1}), \ldots\rangle && \text{by } \langle \mathsf{refl},\ \mathsf{refl} + \tau_{\mathcal{U}}(\mathbf{1}, \mathfrak{f}^{\mathsf{b}}_m(b)), \ldots\rangle \\
=\ &\langle n+m+1,\ \mathfrak{f}^{\mathsf{b}}_n(a) + \mathfrak{f}^{\mathsf{b}}_m(b) + \mathbf{1}, \ldots\rangle && \text{by } \langle \mathsf{refl},\ \alpha^{-1}_{\mathcal{U}}(\mathfrak{f}^{\mathsf{b}}_n(a), \mathfrak{f}^{\mathsf{b}}_m(b), \mathbf{1}), \ldots\rangle \\
\equiv\ &\langle \mathsf{s}, \mathsf{add}, \ldots\rangle\langle n+m,\ \mathfrak{f}^{\mathsf{b}}_n(a) + \mathfrak{f}^{\mathsf{b}}_m(b), \ldots\rangle \\
=\ &\langle \mathsf{s}, \mathsf{add}, \ldots\rangle(\mathfrak{f}_\bullet(\langle n, a\rangle \oplus \langle m, b\rangle)) && \text{by } [\langle \mathsf{s}, \mathsf{add}, \ldots\rangle](h_m(b)) \\
=\ &\mathfrak{f}_\bullet(\langle \mathsf{s}, \mathsf{i}\rangle(\langle n, a\rangle \oplus \langle m, b\rangle)) && \text{by (5.50)} \\
\equiv\ &\mathfrak{f}_\bullet(\langle n+1, \mathsf{i}(a)\rangle \oplus \langle m, b\rangle && \text{by (5.29)},
\end{aligned}$$

  where $\alpha_{\mathcal{U}}$ is the associativity for the coproduct of types (and recall that $n + m + 1 \equiv (n+1) + m$);

- given $n : \mathbb{N}$, $a : \mathsf{del}_n$ and a term h as in (5.52), the requirement relative to the constructor tw of $\mathsf{del}_{(-)}$ in the proof by induction entails unravelling from path algebra the 2-path shown in Fig. 5.10.

The given constructions of $(\mathfrak{f}_\bullet)_0$ and $(\mathfrak{f}_\bullet)_2$ determine a path $(\mathfrak{f}^{\mathsf{b}}_\bullet)_0 : \mathbf{0} = \mathfrak{f}^{\mathsf{b}}_\bullet(\underline{\mathsf{pt}_0})$, given again by reflexivity, and a family

$$(\mathfrak{f}^{\mathsf{b}}_\bullet)_2 : \Pi\,(x, y : \mathsf{del}_\bullet)\,.\,\mathfrak{f}^{\mathsf{b}}_\bullet(x) + \mathfrak{f}^{\mathsf{b}}_\bullet(y) = \mathfrak{f}^{\mathsf{b}}_\bullet(x \oplus y),$$

obtained for every x, y : $\mathsf{del}_\bullet$ by taking the appropriate projection of the path $(\mathfrak{f}_\bullet)_2(x, y)$; in particular,

$$\begin{aligned}
(\mathfrak{f}^{\mathsf{b}}_\bullet)_2(\langle 0, \mathsf{pt}_0\rangle, b) =\ &\lambda_{\mathcal{U}}(\mathfrak{f}^{\mathsf{b}}_m(b)), \\
(\mathfrak{f}^{\mathsf{b}}_\bullet)_2(\langle n+1, \mathsf{i}(a)\rangle, \langle m, b\rangle) =\ &\alpha_{\mathcal{U}}(\mathfrak{f}^{\mathsf{b}}_n(a), \mathbf{1}, \mathfrak{f}^{\mathsf{b}}_m(b)) \cdot (\mathsf{refl} + \tau_{\mathcal{U}}(\mathbf{1}, \mathfrak{f}^{\mathsf{b}}_m(b))) \\
&\cdot \alpha^{-1}_{\mathcal{U}}(\mathfrak{f}^{\mathsf{b}}_n(a), \mathfrak{f}^{\mathsf{b}}_m(b), \mathbf{1}) \cdot ((\mathfrak{f}^{\mathsf{b}}_\bullet)_2(\langle n, a\rangle, \langle m, b\rangle) + \mathsf{refl}).
\end{aligned}$$

The families of 2-paths $(\mathfrak{f}_\bullet)_\alpha$, $(\mathfrak{f}_\bullet)_\lambda$, $(\mathfrak{f}_\bullet)_\rho$ and $(\mathfrak{f}_\bullet)_\tau$ are found, again, by induction. By Lemma 2.143 and Lemma 5.26, we can use connectedness: it is enough to examine the 2-paths where the involved terms are of the form $\langle n, \mathsf{pt}_n\rangle$, and proceed by induction on $n : \mathbb{N}$. These 2-paths are given by families of 2-paths in $\mathbb{N}$ (in the base), which are easy to find, and by families of 2-paths in $\mathcal{U}$. We show in Fig. 5.11 how to derive $(\mathfrak{f}_\bullet)_\alpha$, focussing on the the mentioned 2-paths in $\mathcal{U}$; the other families are found similarly. □

Towards a Proof of the Equivalence

This section is devoted to the search for a proof that $\mathfrak{f}_n : \mathsf{del}_n \to \mathcal{B}\mathsf{S}_n$ is an equivalence for every $n : \mathbb{N}$. In order to reach this goal, we would need to verify that the definition of the indexed HIT $\mathsf{del}_{(-)}$ does indeed reflect the combinatorial structure of finite types. Although the final result rests on the claims in Assumption 5.91, we believe that the strategy we describe is sound and conducive to the construction of the sought equivalence.

A preliminary and easy consideration concerns the functions at the lowest degrees in the family.

Lemma 5.53. *The functions* $\mathfrak{f}_0 : \mathsf{del}_0 \to \mathcal{B}\mathsf{S}_0$ *and* $\mathfrak{f}_1 : \mathsf{del}_1 \to \mathcal{B}\mathsf{S}_1$ *are equivalences.*

Proof. Lemmata 5.8 and 5.25 prove that all the types involved are contractible; hence, by Lemma 2.97, the functions $\mathfrak{f}_0$ and $\mathfrak{f}_1$ are equivalences. □

For a general $n : \mathbb{N}$, showing that $\mathfrak{f}_n$ is an equivalence reveals itself to be a more complex task. The following lemma shows a condition sufficient to prove this claim.

Lemma 5.54. *Let* $n : \mathbb{N}$. *If* $[\mathfrak{f}_n^{\mathsf{b}}] : (a = \mathsf{pt}_n) \to (\mathfrak{f}_n^{\mathsf{b}}(a) = [\![n]\!])$ *has a section for every* $a : \mathsf{del}_n$, *then* $\mathfrak{f}_n$ *is an equivalence, in the sense that there is a function*

$$\left(\Pi\,(a : \mathsf{del}_n)\,.\,\Pi\left(p : \mathfrak{f}_n^{\mathsf{b}}(a) = [\![n]\!]\right).\,\Sigma\,(q : a = \mathsf{pt}_n)\,.\,[\mathfrak{f}_n^{\mathsf{b}}](q) = p\right) \to \mathsf{IsEquiv}(\mathfrak{f}_n).$$

Proof. By Lemma 2.120(ii), $\mathfrak{f}_n$ is an equivalence if its fibers are contractible, i.e., if there is a term in

$$\Pi\,(b : \mathcal{B}\mathsf{S}_n)\,.\,\mathsf{IsContr}(\mathsf{fib}_{\mathfrak{f}_n}(b)). \tag{5.55}$$

Since $\mathcal{B}\mathsf{S}_n$ is connected (Lemma 5.9) and $\mathsf{IsContr}(\mathsf{fib}_{\mathfrak{f}_n}(b))$ is a (-1)-type for every $b : \mathcal{B}\mathsf{S}_n$ (Lemma 2.77), then, by Lemma 2.143, a function term of the type in (5.55) can be obtained given a term in $\mathsf{IsContr}(\mathsf{fib}_{\mathfrak{f}_n}(b))$ for any one choice of $b : \mathcal{B}\mathsf{S}_n$, such as $b :\equiv \mathfrak{f}_n(\mathsf{pt}_n)$. That is, we can infer that $\mathfrak{f}_n$ is an equivalence if we prove

$$\mathsf{IsContr}(\mathsf{fib}_{\mathfrak{f}_n}(\mathfrak{f}_n(\mathsf{pt}_n))). \tag{5.56}$$

We can construct a function

$$\mathsf{IsContr}(\mathsf{fib}_{\mathfrak{f}_n^{\mathsf{b}}}([\![n]\!])) \to \mathsf{IsContr}(\mathsf{fib}_{\mathfrak{f}_n}(\mathfrak{f}_n(\mathsf{pt}_n))), \tag{5.57}$$

as follows. Assuming $\mathsf{IsContr}(\Sigma\,(a : \mathsf{del}_n)\,.\,\mathfrak{f}_n^{\mathsf{b}}(a) = [\![n]\!])$ gives us a center of contraction $\langle a_0, p_0\rangle$ with $a_0 : \mathsf{del}_n$ and $p_0 : \mathfrak{f}_n^{\mathsf{b}}(a_0) = [\![n]\!]$, and a proof of contraction

$c : \Pi\left(s : \Sigma\,(a : \mathsf{del}_n)\,.\,\mathfrak{f}_n^{\mathsf{b}}(a) = [\![n]\!]\right).\langle a_0, p_0\rangle = s$, which can be decomposed for base and fibers of the Σ-type, as:

$$c^{\mathsf{b}} : \Pi\,(a : \mathsf{del}_n)\,.\,(\mathfrak{f}_n^{\mathsf{b}}(a) = [\![n]\!]) \to (a_0 = a),$$
$$c^{\mathsf{f}} : \Pi\,(a : \mathsf{del}_n)\,.\,\Pi\left(p : \mathfrak{f}_n^{\mathsf{b}}(a) = [\![n]\!]\right).\left(c^{\mathsf{b}}(a,p)\right)_*^{(z\mapsto \mathfrak{f}_n^{\mathsf{b}}(z)=[\![n]\!])}(p_0) = p,$$

by defining $c^{\mathsf{b}}(a,p) :\equiv \overline{\mathsf{pr}_1}(c\langle a,p\rangle)$ and $c^{\mathsf{f}}(a,p) :\equiv \overline{\mathsf{pr}_2}(c\langle a,p\rangle)$. In order to prove $\mathsf{IsContr}(\Sigma\,(a : \mathsf{del}_n)\,.\,\mathfrak{f}_n(a) = \mathfrak{f}_n(\mathsf{pt}_n))$ we need:

- a center of contraction, which is given by $\langle a_0, \langle p_0, \ldots\rangle\rangle$, using the shorthand notation of Remark 2.78;
- a proof of contraction, i.e., for every $a : \mathsf{del}_n$ and $p : \mathfrak{f}_n(a) = \mathfrak{f}_n(\mathsf{pt}_n)$, a path

  $$\langle a_0, \langle p_0, \ldots\rangle\rangle = \langle a, p\rangle.$$

  Using Lemma 2.66, we can obtain such a path componentwise. A path $a_0 = a$ is given by $c^{\mathsf{b}}(a, \overline{\mathsf{pr}_1}(p))$. As for the second component, it is enough to provide a path

  $$\overline{\mathsf{pr}_1}\left(\left(c^{\mathsf{b}}(a, \overline{\mathsf{pr}_1}(p))\right)_*^{(z\mapsto \mathfrak{f}_n(z)=\mathfrak{f}_n(\mathsf{pt}_n))}\langle p_0, \ldots\rangle\right) = \overline{\mathsf{pr}_1}(p),$$

  which can be given by

  $$\begin{aligned}
  &\overline{\mathsf{pr}_1}\left(\left(c^{\mathsf{b}}(a, \overline{\mathsf{pr}_1}(p))\right)_*^{(z\mapsto \mathfrak{f}_n(z)=\mathfrak{f}_n(\mathsf{pt}_n))}\langle p_0, \ldots\rangle\right) \\
  &\quad = \left(c^{\mathsf{b}}(a, \overline{\mathsf{pr}_1}(p))\right)_*^{(z\mapsto \mathfrak{f}_n^{\mathsf{b}}(z)=\mathfrak{f}_n^{\mathsf{b}}(\mathsf{pt}_n))}(p_0) && \text{by path algebra} \\
  &\quad = \overline{\mathsf{pr}_1}(p) && \text{by } c^{\mathsf{f}}(a, \overline{\mathsf{pr}_1}(p)).
  \end{aligned}$$

By what above, we conclude that we can prove that $\mathfrak{f}_n$ is an equivalence by showing that the type $\mathsf{fib}_{\mathfrak{f}_n^{\mathsf{b}}}([\![n]\!])$ is contractible. Using $\langle \mathsf{pt}_n, \mathsf{refl}_{[\![n]\!]}\rangle$ as center of contraction, the proof of contraction requires providing, for every $a : \mathsf{del}_n$ and $p : \mathfrak{f}_n^{\mathsf{b}}(a) = [\![n]\!]$, a path $\langle \mathsf{pt}_n, \mathsf{refl}_{[\![n]\!]}\rangle = \langle a, p\rangle$, i.e., by Lemmata 2.66 and 2.102, a term in

$$\Sigma\,(q : a = \mathsf{pt}_n)\,.\,[\mathfrak{f}_n^{\mathsf{b}}](q) = p.$$

In other words, to obtain the sought proof of contraction, it is enough to show that the function

$$[\mathfrak{f}_n^{\mathsf{b}}] : (a = \mathsf{pt}_n) \to (\mathfrak{f}_n^{\mathsf{b}}(a) = [\![n]\!])$$

has a section. □

A strategy to prove that $\mathfrak{f}_n$ is an equivalence for every n, and hence that the types in the families del and $\mathcal{BS}$ are termwise equivalent, is then to find a term

$$\mathfrak{e} : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : \mathsf{del}_n)\,.\,\Pi\left(p : \mathfrak{f}^{\mathsf{b}}_n(a) = [\![n]\!]\right).\,\Sigma\,(q : a = \mathsf{pt}_n)\,.\,[\mathfrak{f}^{\mathsf{b}}_n](q) = p. \tag{5.58}$$

The following paragraphs present our approach to obtain such a term. As mentioned, the final step of our proof relies on a lemma on combinatorics of finite types which has not been formalized (Assumption 5.91).

A close look at the type in (5.58) suggests a possible definition of the term $\mathfrak{e}$ employing the elimination principle of $\mathsf{del}_{(-)}$. The first observation we might make is that the family

$$\left(n \mapsto \left(a \mapsto \Pi\left(p : \mathfrak{f}^{\mathsf{b}}_n(a) = [\![n]\!]\right).\,\Sigma\,(q : a = \mathsf{pt}_n)\,.\,[\mathfrak{f}^{\mathsf{b}}_n](q) = p\right)\right) \;: \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \to \mathcal{U}$$

is a family of 0-types. Indeed, using Remark 2.75, we can verify this by showing that, for every $n : \mathbb{N}$, $a : \mathsf{del}_n$ and $p : \mathfrak{f}^{\mathsf{b}}_n(a) = [\![n]\!]$, both the identity type $a = \mathsf{pt}_n$ and all the types in the family $(q \mapsto [\mathfrak{f}^{\mathsf{b}}_n](q) = p)$ are 0-types (the former by T_{del}, the latter is a (-1)-type by Lemma 5.6). Hence, a term $\mathfrak{e}$ as in (5.58) can be constructed by means of the elimination principle of $\mathsf{del}_{(-)}$, if we provide terms:

$$\mathfrak{e}'_0 : \Pi\,(p : [\![0]\!] = [\![0]\!])\,.\,\Sigma\,(q : \mathsf{pt}_0 = \mathsf{pt}_0)\,.\,[\mathfrak{f}^{\mathsf{b}}_0](q) = p, \tag{5.59}$$

$$\begin{aligned}\mathfrak{e}'_{\mathsf{i}} : {}& \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : \mathsf{del}_n)\,.\\ & \left(\Pi\left(p : \mathfrak{f}^{\mathsf{b}}_n(a) = [\![n]\!]\right).\,\Sigma\,(q : a = \mathsf{pt}_n)\,.\,[\mathfrak{f}^{\mathsf{b}}_n](q) = p\right) \to\\ & \Pi\left(p : \mathfrak{f}^{\mathsf{b}}_{n+1}(\mathsf{i}(a)) = [\![n+1]\!]\right).\,\Sigma\,(q : \mathsf{i}(a) = \mathsf{pt}_{n+1})\,.\,[\mathfrak{f}^{\mathsf{b}}_{n+1}](q) = p,\end{aligned} \tag{5.60}$$

$$\begin{aligned}\mathfrak{e}'_{\mathsf{tw}} : {}& \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : \mathsf{del}_n)\,.\\ & \Pi\left(a' : \Pi\left(p : \mathfrak{f}^{\mathsf{b}}_n(a) = [\![n]\!]\right).\,\Sigma\,(q : a = \mathsf{pt}_n)\,.\,[\mathfrak{f}^{\mathsf{b}}_n](q) = p\right).\\ & (\mathsf{tw}_a)^{E_n}_*\left(\mathfrak{e}'_{\mathsf{i}}(n+1, \mathsf{i}(a), \mathfrak{e}'_{\mathsf{i}}(n, a, a'))\right) = \mathfrak{e}'_{\mathsf{i}}(n+1, \mathsf{i}(a), \mathfrak{e}'_{\mathsf{i}}(n, a, a')),\end{aligned} \tag{5.61}$$

where $E : \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_{n+2} \to \mathcal{U}$ in the type of $\mathfrak{e}'_{\mathsf{tw}}$ is defined as

$$E_n :\equiv \left(z \mapsto \Pi\left(p : \mathfrak{f}^{\mathsf{b}}_{n+2}(z) = [\![n+2]\!]\right).\,\Sigma\,(q : z = \mathsf{pt}_{n+2})\,.\,[\mathfrak{f}^{\mathsf{b}}_{n+2}](q) = p\right) \tag{5.62}$$

for every $n : \mathbb{N}$. Using techniques similar to those described in Lemma 5.37, we can examine each of the three requirements separately on base and fibers of Σ-types; these are treated in the following lemmata, which we will prove later in this section (Lemma 5.67 will require Assumption 5.91).

Lemma 5.63 (relevant to (5.59))**.** *For every $p : [\![0]\!] = [\![0]\!]$, we construct terms*

$$\mathfrak{e}^{\mathsf{b}}_0(p) : \mathsf{pt}_0 = \mathsf{pt}_0 \qquad \textit{and} \qquad \mathfrak{e}^{\mathsf{f}}_0(p) : [\mathfrak{f}^{\mathsf{b}}_0](\mathfrak{e}^{\mathsf{b}}_0(p)) = p.$$

Lemma 5.64 (relevant to (5.60))**.** *Let $n : \mathbb{N}$ and $a : \mathsf{del}_n$. Given terms*

$$\mathfrak{e}^{\mathsf{b}}_{n,a} : (\mathfrak{f}^{\mathsf{b}}_n(a) = [\![n]\!]) \to (a = \mathsf{pt}_n) \quad \textit{and} \tag{5.65}$$

$$\mathfrak{e}^{\mathsf{f}}_{n,a} : \Pi\left(p : \mathfrak{f}^{\mathsf{b}}_n(a) = [\![n]\!]\right).\, [\mathfrak{f}^{\mathsf{b}}_n](\mathfrak{e}^{\mathsf{b}}_{n,a}(p)) = p, \tag{5.66}$$

we construct functions

$$\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathfrak{e}^{\mathsf{b}}_{n,a}) : (\mathfrak{f}^{\mathsf{b}}_{n+1}(\mathsf{i}(a)) = [\![n+1]\!]) \to (\mathsf{i}(a) = \mathsf{pt}_{n+1}) \quad \textit{and}$$

$$\mathfrak{e}^{\mathsf{f}}_{n+1,\mathsf{i}(a)}(\mathfrak{e}^{\mathsf{b}}_{n,a}, \mathfrak{e}^{\mathsf{f}}_{n,a}) : \Pi\left(p : \mathfrak{f}^{\mathsf{b}}_{n+1}(\mathsf{i}(a)) = [\![n+1]\!]\right).\, [\mathfrak{f}^{\mathsf{b}}_{n+1}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathfrak{e}^{\mathsf{b}}_{n,a}, p)) = p.$$

Lemma 5.67 (relevant to (5.61))**.** *Let $n : \mathbb{N}$ and $p : [\![n+2]\!] = [\![n+2]\!]$. Given a term $\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}$ as in (5.65) for $a :\equiv \mathsf{pt}_n$, there is a 2-path*

$$\begin{aligned} &\mathfrak{e}^{\mathsf{b}}_{n+2,\mathsf{pt}_{n+2}}(\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}),\ \gamma_{[\![n]\!]} \cdot p) \\ = \mathsf{tw}_{\mathsf{pt}_n} \cdot\ &\mathfrak{e}^{\mathsf{b}}_{n+2,\mathsf{pt}_{n+2}}(\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}),\ p), \end{aligned} \tag{5.68}$$

with $\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}$ (and $\mathfrak{e}^{\mathsf{b}}_{n+2,\mathsf{pt}_{n+2}}$) defined in Lemma 5.64.

Assuming the lemmata above, we are ready to prove the main result of this section.

Theorem 5.69. *The function $\mathfrak{f}_n : \mathsf{del}_n \to \mathcal{BS}_n$ is an equivalence for every $n : \mathbb{N}$. Hence, there is a symmetric monoidal equivalence $\mathsf{del}_\bullet \simeq \mathcal{BS}_\bullet$.*

Proof. By Lemma 5.54, it is enough to find a term $\mathfrak{e}$ as in (5.58); we proceed by the elimination principle of $\mathsf{del}_{(-)}$:

- a term $\mathfrak{e}'_0$ as in (5.59) is given by $\mathfrak{e}'_0(p) :\equiv \langle \mathfrak{e}^{\mathsf{b}}_0(p), \mathfrak{e}^{\mathsf{f}}_0(p)\rangle$, with $\mathfrak{e}^{\mathsf{b}}_0$ and $\mathfrak{e}^{\mathsf{f}}_0$ defined in the proof of Lemma 5.63;
- the term $\mathfrak{e}'_{\mathsf{i}}$ in (5.60) is given by

  $$\mathfrak{e}'_{\mathsf{i}}(n, a, a', p) :\equiv \langle \mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathsf{pr}_1(a'), p),\ \mathfrak{e}^{\mathsf{f}}_{n+1,\mathsf{i}(a)}(\mathsf{pr}_1(a'), \mathsf{pr}_2(a'), p)\rangle,$$

  with $\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}$ and $\mathfrak{e}^{\mathsf{f}}_{n+1,\mathsf{i}(a)}$ defined for every $n : \mathbb{N}$ and $a : \mathsf{del}_n$ in the proof of Lemma 5.64;
- as for the term $\mathfrak{e}'_{\mathsf{tw}}$ in (5.61), the 2-path constructed in the proof of Lemma 5.67 is sufficient. Indeed, given $n : \mathbb{N}$, $a : \mathsf{del}_n$ and a term

  $$a' : \Pi\left(p : \mathfrak{f}^{\mathsf{b}}_n(a) = [\![n]\!]\right).\, \Sigma\,(q : a = \mathsf{pt}_n).\, [\mathfrak{f}^{\mathsf{b}}_n](q) = p,$$

  we have that a term

  $$(\mathsf{tw}_a)^{E_n}_*\left(\mathfrak{e}'_{\mathsf{i}}(n+1, \mathsf{i}(a), \mathfrak{e}'_{\mathsf{i}}(n, a, a'))\right) = \mathfrak{e}'_{\mathsf{i}}(n+1, \mathsf{i}(a), \mathfrak{e}'_{\mathsf{i}}(n, a, a')),$$

for $\mathfrak{e}'_{\mathsf{i}}$ as in (5.60), can be obtained, by function extensionality, from a family of terms

$$(\mathsf{tw}_a)^{E_n}_* \left(\mathfrak{e}'_{\mathsf{i}}(n+1, \mathsf{i}(a), \mathfrak{e}'_{\mathsf{i}}(n, a, a'), p)\right) = \mathfrak{e}'_{\mathsf{i}}(n+1, \mathsf{i}(a), \mathfrak{e}'_{\mathsf{i}}(n, a, a'), p)$$

for every $p : \mathfrak{f}^{\mathsf{b}}_{n+2}(\mathsf{i}(\mathsf{i}(a))) = [\![n+2]\!]$. This is a family of 2-paths in a Σ-type whose fibers are 1-types, so it is enough to provide a 2-path in the base, i.e, to find, for every p as above, a term

$$(\mathsf{tw}_a)^{E^{\mathsf{b}}_n}_* \left(\mathfrak{e}^{\mathsf{b}}_{n+2,\mathsf{i}(\mathsf{i}(a))}(\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathsf{pr}_1(a')), p)\right) = \mathfrak{e}^{\mathsf{b}}_{n+2,\mathsf{i}(\mathsf{i}(a))}(\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathsf{pr}_1(a')), p),$$

where the family E^{b} is defined by $E^{\mathsf{b}}_n :\equiv \left(z \mapsto (\mathfrak{f}^{\mathsf{b}}_{n+2}(z) = [\![n+2]\!]) \to (z = \mathsf{pt}_{n+2})\right)$. Transporting in such a family of functions entails finding a term

$$\mathfrak{e}^{\mathsf{b}}_{n+2,\mathsf{i}(\mathsf{i}(a))}(\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathsf{pr}_1(a')), (\mathsf{tw}_a^{-1})^{E^{\mathsf{b}\prime}_n}_*(p)) = \mathsf{tw}_a \cdot \mathfrak{e}^{\mathsf{b}}_{n+2,\mathsf{i}(\mathsf{i}(a))}(\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathsf{pr}_1(a')), p),$$

with $E^{\mathsf{b}\prime}_n :\equiv (z \mapsto (\mathfrak{f}^{\mathsf{b}}_{n+2}(z) = [\![n+2]\!]))$. Path algebra concerning transport in families of paths further simplifies the goal above to

$$\mathfrak{e}^{\mathsf{b}}_{n+2,\mathsf{i}(\mathsf{i}(a))}(\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathsf{pr}_1(a')), [\mathfrak{f}^{\mathsf{b}}_{n+2}](\mathsf{tw}_a) \cdot p) = \mathsf{tw}_a \cdot \mathfrak{e}^{\mathsf{b}}_{n+2,\mathsf{i}(\mathsf{i}(a))}(\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathsf{pr}_1(a')), p),$$

and, after an application of the computation rule (5.42), to the 2-path

$$\mathfrak{e}^{\mathsf{b}}_{n+2,\mathsf{i}(\mathsf{i}(a))}(\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathsf{pr}_1(a')), \gamma_{\mathfrak{f}^{\mathsf{b}}_n(a)} \cdot p) = \mathsf{tw}_a \cdot \mathfrak{e}^{\mathsf{b}}_{n+2,\mathsf{i}(\mathsf{i}(a))}(\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathsf{pr}_1(a')), p).$$

Such a family of 2-paths (parametrized by $a : \mathsf{del}_n$) can be obtained from the one in (5.68) using Lemma 2.143, as the type of 2-paths in a 1-type is a (-1)-type and del_n is connected for every $n : \mathbb{N}$ (Lemma 5.26).

Then $\mathfrak{f}_n : \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \to \mathcal{BS}_n$ is an equivalence for every $n : \mathbb{N}$. By Remark 2.99, the function $\mathfrak{f}_\bullet \equiv \langle \mathsf{id}_{\mathbb{N}}, \mathfrak{f}\rangle : \mathsf{del}_\bullet \to \mathcal{BS}_\bullet$ from Definition 5.48 is also an equivalence; this can then be promoted to a symmetric monoidal equivalence, by virtue of Theorem 5.51 and Remark 4.9. □

Proofs of Lemmata

The remaining part of this section is devoted to the proof of Lemmata 5.63, 5.64 and 5.67; in particular, Lemma 5.67 is where we use Assumption 5.91. The proof of the first lemma is immediate.

Proof of Lemma 5.63. For every path $p : \mathbf{0} = \mathbf{0}$, we define $\mathfrak{e}^{\mathsf{b}}_0(p) :\equiv \mathsf{refl}_{\mathsf{pt}_0} : \mathsf{pt}_0 = \mathsf{pt}_0$ (note that the path does not actually depend on p). A term $\mathfrak{e}^{\mathsf{f}}_0 : [\mathfrak{f}^{\mathsf{b}}_0](\mathfrak{e}^{\mathsf{b}}_0(p)) = p$,

whose type is judgmentally equal to $\mathsf{refl}_{\mathbf{0}} = p$, is obtained by the fact that the identity type $\mathbf{0} = \mathbf{0}$ is equivalent to the type of equivalences $\mathbf{0} \simeq \mathbf{0}$ by univalence, and to $\mathbf{0} \to \mathbf{0}$ by Lemma 2.120; this is a (-1)-type by Remark 2.75. □

Lemma 5.64 is more challenging and requires some results about the combinatorics of finite types.

An equivalence $e : A \simeq [\![n]\!]$ can be promoted to an equivalence $\mathsf{incr}(e) : A + \mathbf{1} \simeq [\![n+1]\!]$, which maps $\mathsf{inr}(*) : A + \mathbf{1}$ to $\mathsf{inr}(*) : [\![n+1]\!]$, and whose restriction to A is e itself (Corollary 2.124). In the opposite direction, an equivalence $A + \mathbf{1} \simeq [\![n+1]\!]$ can be "adjusted" to one sending $\mathsf{inr}(*) : A + \mathbf{1}$ to $\mathsf{inr}(*) : [\![n+1]\!]$, which can be then restricted to A. The ensuing map $(A + \mathbf{1} \simeq [\![n+1]\!]) \to (A \simeq [\![n]\!])$ is obviously not an equivalence. Nonetheless, understanding its properties will be useful in the pursue of the term $\mathfrak{e}$ in (5.58), where the elimination principle of $\mathsf{del}_{(-)}$ asks us to reason about certain paths between finite types of cardinality $n+1$, recursively, in terms of paths between finite types of cardinality n. Since, ultimately, we want to show that the family of functions $\mathfrak{f}^{\mathsf{b}}_{(-)}$ establishes a correspondence between such paths and identities in the types of the family $\mathsf{del}_{(-)}$ (i.e., that $\mathfrak{f}_n$ is an *embedding* for every $n : \mathbb{N}$), we will prefer to reason in terms of paths rather than equivalences. However, relations between (finite) types are often more easily described by equivalences than by identities, so we will largely use "univalence algebra" (Section 2.5) to switch between notions.

The recursive definition of $\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}$ in terms of $\mathfrak{e}^{\mathsf{b}}_{n,a}$ is schematically depicted in Fig. 5.6. A path

$$p : \mathfrak{f}^{\mathsf{b}}_{n+1}(\mathsf{i}(a)) = [\![n+1]\!]$$

corresponds to an equivalence

$$\mathsf{ua}^{-1}(p) : \mathfrak{f}^{\mathsf{b}}_{n+1}(\mathsf{i}(a)) \simeq [\![n+1]\!],$$

to which we can compose a symmetry $\sigma : [\![n+1]\!] \simeq [\![n+1]\!]$ and obtain an equivalence fixing the term $\mathsf{inr}(*)$. This can be restricted to an equivalence $\mathfrak{f}^{\mathsf{b}}_n(a) \simeq [\![n]\!]$, corresponding to a path $\overline{p} : \mathfrak{f}^{\mathsf{b}}_n(a) = [\![n]\!]$ which, in turn, can be given as argument to the function $\mathfrak{e}^{\mathsf{b}}_{n,a}$ to produce a path

$$\mathfrak{e}^{\mathsf{b}}_{n,a}(\overline{p}) : a = \mathsf{pt}_n.$$

A single application of $[\mathsf{i}]$ yields a path

$$[\mathsf{i}]\left(\mathfrak{e}^{\mathsf{b}}_{n,a}(\overline{p})\right) : \mathsf{i}(a) = \mathsf{pt}_{n+1},$$

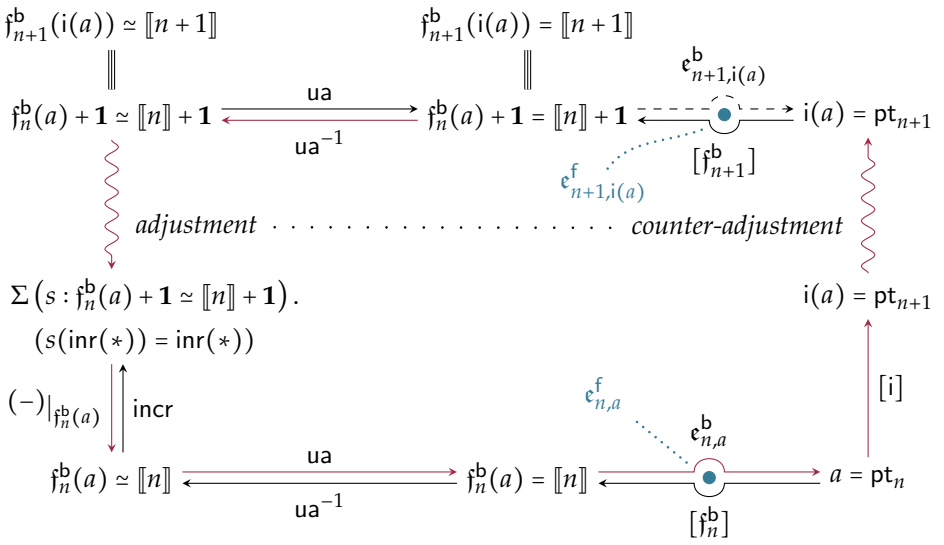


Figure 5.6: Recursive definition of $\epsilon^{\mathsf{b}}_{n+1,\mathsf{i}(a)}$ (dashed arrow) in terms of $\epsilon^{\mathsf{b}}_{n,a}$, as the composition of the red arrows. In order to satisfy the requirement given by $\epsilon^{\mathsf{f}}_{n+1,\mathsf{i}(a)}$, the composition of $[\mathfrak{f}^{\mathsf{b}}_{n+1}]$ after the red arrows needs to be the identity.

which is already the target type of $\epsilon^{\mathsf{b}}_{n+1,\mathsf{i}(a)}$; however, in order to fulfill the constraints of $\epsilon^{\mathsf{f}}_{n+1,\mathsf{i}(a)}$, we will need to concatenate to $[\mathsf{i}](\epsilon^{\mathsf{b}}_{n,a}(\overline{p}))$ a loop $\mathsf{pt}_{n+1} = \mathsf{pt}_{n+1}$ reversing the symmetry σ with which we modified the original equivalence.

We will start this endeavour by defining the symmetries used in the process described above (wavy arrows in Fig. 5.6).

Definition 5.70. A function $\mathfrak{m} : \Pi\,(n : \mathbb{N})\,.\,[\![n]\!] \to ([\![n]\!] = [\![n]\!])$ is defined inductively on n and on $i : [\![n]\!]$, as follows:

$$\begin{aligned} \mathfrak{m}_{n+1}(\mathsf{inr}(*)) &:\equiv \mathsf{refl}_{[\![n+1]\!]}, \\ \mathfrak{m}_{n+2}(\mathsf{inl}(\mathsf{inr}(*))) &:\equiv \gamma_{[\![n]\!]}, \\ \mathfrak{m}_{n+3}(\mathsf{inl}(\mathsf{inl}(i))) &:\equiv \gamma_{[\![n+1]\!]} \cdot [\mathsf{add}](\mathfrak{m}_{n+2}(\mathsf{inl}(i))) \cdot \gamma_{[\![n+1]\!]} \end{aligned} \tag{5.71}$$

for every $i : [\![n+1]\!]$; all the other cases are obtained *ex falso* (namely, $\mathfrak{m}_0(i)$, $\mathfrak{m}_1(\mathsf{inl}(i))$ and $\mathfrak{m}_2(\mathsf{inl}(\mathsf{inl}(i)))$ for $i : \mathbf{0}$). We note that, for every $i : [\![n+1]\!] \equiv [\![n]\!] + \mathbf{1}$, the loop $\mathfrak{m}_{n+1}(i) : [\![n+1]\!] = [\![n+1]\!]$ corresponds, via univalence and (2.129), to the transposition of $\mathsf{inr}(*)$ and i, realized via a composition of ω's. An example is provided in Fig. 5.7.

Lemma 5.72. *For every $n : \mathbb{N}$ and $i : [\![n]\!]$, the path $\mathfrak{m}_n(i)$ is its own inverse, i.e., $(\mathfrak{m}_n(i))^{-1} = \mathfrak{m}_n(i)$. Hence, $\mathsf{ua}^{-1}(\mathfrak{m}_n(i) \cdot \mathfrak{m}_n(i)) = \mathsf{id}_{[\![n]\!]}$.*

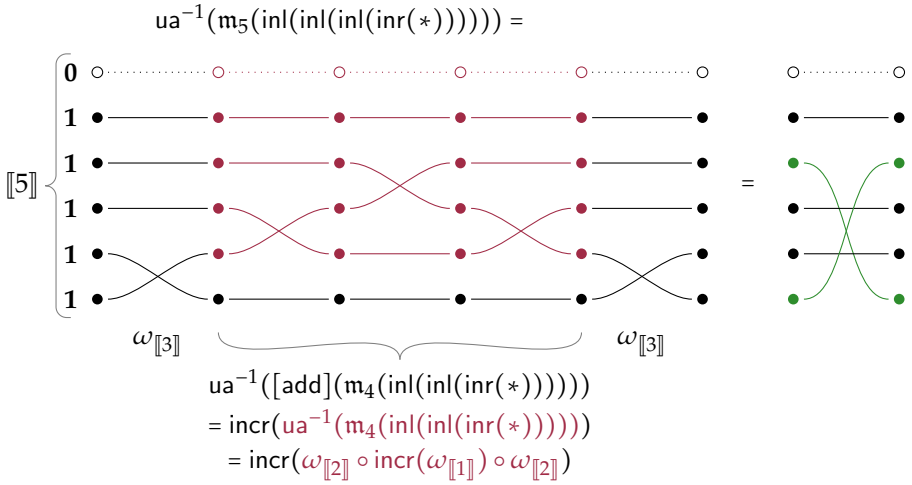


Figure 5.7: Example of application of the function $\mathfrak{m}$. The type $[\![5]\!]$ computes to the canonical finite type $[5]$; the equivalence $\mathsf{ua}^{-1}(\mathfrak{m}_5(\mathsf{inl}(\mathsf{inl}(\mathsf{inl}(\mathsf{inr}(*))))))$ has, as underlying function, the transposition of the terms $\mathsf{inr}(*)$ and $\mathsf{inl}(\mathsf{inl}(\mathsf{inl}(\mathsf{inr}(*))))$: $[5]$ (in green, on the right). The recursive call is shown in red.

Proof. The claim is proved by induction on n and i, following (5.71); we will show the recursive case, the other ones being easily deduced. For $n : \mathbb{N}$ and $i : [\![n+1]\!]$, we have:

$$\begin{aligned}
(\mathfrak{m}_{n+3}(\mathsf{inl}(\mathsf{inl}(i))))^{-1} &\equiv (\gamma_{[\![n+1]\!]} \cdot [\mathsf{add}](\mathfrak{m}_{n+2}(\mathsf{inl}(i))) \cdot \gamma_{[\![n+1]\!]})^{-1} \\
&= \gamma^{-1}_{[\![n+1]\!]} \cdot [\mathsf{add}]((\mathfrak{m}_{n+2}(\mathsf{inl}(i)))^{-1}) \cdot \gamma^{-1}_{[\![n+1]\!]} && \text{by path algebra} \\
&= \gamma^{-1}_{[\![n+1]\!]} \cdot [\mathsf{add}](\mathfrak{m}_{n+2}(\mathsf{inl}(i))) \cdot \gamma^{-1}_{[\![n+1]\!]} && \text{inductively} \\
&= \gamma_{[\![n+1]\!]} \cdot [\mathsf{add}](\mathfrak{m}_{n+2}(\mathsf{inl}(i))) \cdot \gamma_{[\![n+1]\!]} && \text{by Remark 5.16} \\
&\equiv \mathfrak{m}_{n+3}(\mathsf{inl}(\mathsf{inl}(i))).
\end{aligned}$$

□

Lemma 5.73. *For every $n : \mathbb{N}$ and $i : [\![n+1]\!] \equiv [\![n]\!] + \mathbf{1}$, there is a path*

$$\mathsf{ua}^{-1}(\mathfrak{m}_{n+1}(i))\ (i) =_{[\![n+1]\!]} \mathsf{inr}(*).$$

Proof. Following (5.71), we proceed by induction on n and i:

- for every $n : \mathbb{N}$, we have

$$\begin{aligned}
&\mathsf{ua}^{-1}(\mathfrak{m}_{n+1}(\mathsf{inr}(*)))\ (\mathsf{inr}(*)) \\
&\qquad \equiv \mathsf{ua}^{-1}(\mathsf{refl})\ (\mathsf{inr}(*)) \equiv \mathsf{id}(\mathsf{inr}(*)) \equiv \mathsf{inr}(*);
\end{aligned}$$

- for every $n : \mathbb{N}$, we have

$$\begin{aligned}&\mathsf{ua}^{-1}(\mathfrak{m}_{n+2}(\mathsf{inl}(\mathsf{inr}(*)))) \; (\mathsf{inl}(\mathsf{inr}(*)))\\ &\quad \equiv (\mathsf{ua}^{-1}(\gamma_{[\![n]\!]})) \; (\mathsf{inl}(\mathsf{inr}(*))) = \omega_{[\![n]\!]}(\mathsf{inl}(\mathsf{inr}(*))) \equiv \mathsf{inr}(*);\end{aligned}$$

- for every $n : \mathbb{N}$, assuming as inductive hypothesis that

$$\mathsf{ua}^{-1}(\mathfrak{m}_{n+2}(j)) \; (j) = \mathsf{inr}(*) \quad \text{for any } j : [\![n+2]\!], \tag{5.74}$$

we have, for every $i : [\![n+1]\!]$,

$$\begin{aligned}&\mathsf{ua}^{-1}(\mathfrak{m}_{n+3}(\mathsf{inl}(\mathsf{inl}(i)))) \; (\mathsf{inl}(\mathsf{inl}(i)))\\ &\quad \equiv \mathsf{ua}^{-1}(\gamma_{[\![n+1]\!]} \cdot [\mathsf{add}](\mathfrak{m}_{n+2}(\mathsf{inl}(i))) \cdot \gamma_{[\![n+1]\!]}) \; (\mathsf{inl}(\mathsf{inl}(i)))\\ &\quad = \mathsf{ua}^{-1}(\gamma_{[\![n+1]\!]}) \; (\mathsf{ua}^{-1}([\mathsf{add}](\mathfrak{m}_{n+2}(\mathsf{inl}(i)))) \; (\mathsf{ua}^{-1}(\gamma_{[\![n+1]\!]}) \; (\mathsf{inl}(\mathsf{inl}(i)))))\\ &\quad = \omega_{[\![n+1]\!]}(\mathsf{ua}^{-1}([\mathsf{add}](\mathfrak{m}_{n+2}(\mathsf{inl}(i)))) \; (\omega_{[\![n+1]\!]} \; (\mathsf{inl}(\mathsf{inl}(i)))))\\ &\quad \equiv \omega_{[\![n+1]\!]}(\mathsf{ua}^{-1}([\mathsf{add}](\mathfrak{m}_{n+2}(\mathsf{inl}(i)))) \; (\mathsf{inl}(\mathsf{inl}(i))))\\ &\quad = \omega_{[\![n+1]\!]}(\mathsf{incr}(\mathsf{ua}^{-1}(\mathfrak{m}_{n+2}(\mathsf{inl}(i)))) \; (\mathsf{inl}(\mathsf{inl}(i)))) && \text{by (2.129)}\\ &\quad \equiv \omega_{[\![n+1]\!]}(\mathsf{inl}(\mathsf{ua}^{-1}(\mathfrak{m}_{n+2}(\mathsf{inl}(i))) \; (\mathsf{inl}(i))))\\ &\quad = \omega_{[\![n+1]\!]}(\mathsf{inl}(\mathsf{inr}(*))) \equiv \mathsf{inr}(*) && \text{by (5.74).}\end{aligned}$$

All other cases are obtained *ex falso*. □

Similarly, one can prove that $\mathsf{ua}^{-1}(\mathfrak{m}_{n+1}(i)) \; (\mathsf{inr}(*)) = i$ for every $i : [\![n+1]\!]$, and that $\mathsf{ua}^{-1}(\mathfrak{m}_{n+1}(i)) \; (j) = j$ if $(j = i) \to \mathbf{0}$ and $(j = \mathsf{inr}(*)) \to \mathbf{0}$, for every i, j as above.

Remark 5.75. If $p : A + \mathbf{1} = [\![n+1]\!]$, for some $A : \mathcal{U}$ and $n : \mathbb{N}$, we will use the notation

$$\overset{\bullet}{p} :\equiv \quad \mathsf{ua}^{-1}(p) \; (\mathsf{inr}(*)) \quad : [\![n+1]\!].$$

For $p : A + \mathbf{1} + \mathbf{1} = [\![n+2]\!]$, we will also use the notation

$$\overset{\bullet\bullet}{p} :\equiv \; \mathsf{ua}^{-1}(p) \; (\mathsf{inl}(\mathsf{inr}(*))) \; : [\![n+2]\!].$$

Note that $(\gamma_A \cdot p)^{\bullet} = \mathsf{ua}^{-1}(p) \; (\omega_A(\mathsf{inr}(*))) \equiv \overset{\bullet\bullet}{p}$ and, similarly $(\gamma_A \cdot p)^{\bullet\bullet} = \overset{\bullet}{p}$. As an immediate consequence of Lemma 5.73, we have that

$$\mathsf{ua}^{-1}(p \cdot \mathfrak{m}_{n+1}(\overset{\bullet}{p})) \; (\mathsf{inr}(*)) = \mathsf{ua}^{-1}(\mathfrak{m}_{n+1}(\overset{\bullet}{p})) \; (\overset{\bullet}{p}) = \mathsf{inr}(*), \tag{5.76}$$

i.e., the definition of $\mathfrak{m}$ provides us with a suitable path to concatenate to any path $p : A + \mathbf{1} = [\![n+1]\!]$, so that the corresponding equivalence sends $\mathsf{inr}(*) : A + \mathbf{1}$ to $\mathsf{inr}(*) : [\![n+1]\!]$. This motivates the following definition.

Definition 5.77. For $n : \mathbb{N}$, $A : \mathcal{U}$ and $p : A + \mathbf{1} = [\![n+1]\!]$, we define the *reduction* $\overline{p} : A = [\![n]\!]$ of p as the following path:

$$\overline{p} :\equiv \mathsf{ua}\left(\left(\mathsf{ua}^{-1}(p \cdot \mathfrak{m}_{n+1}(\overset{\bullet}{p}))\right)|_A\right),$$

where Corollary 2.124 and (5.76) guarantee that the restriction is well-defined.

The function $(p \mapsto \overline{p})$ is related to $[\mathsf{add}](-)$ in the ways described in the following lemmata.

Lemma 5.78. *For every $n : \mathbb{N}$, $A : \mathcal{U}$ and $p : A = [\![n]\!]$, there is a 2-path $\overline{[\mathsf{add}](p)} = p$, i.e., $(p \mapsto \overline{p})$ is a retraction of $[\mathsf{add}](-)$.*

Proof. The following chain of identities proves the claim:

$$\begin{aligned}
\overline{[\mathsf{add}](p)} &\equiv \mathsf{ua}\left(\left(\mathsf{ua}^{-1}([\mathsf{add}](p) \cdot \mathfrak{m}_{n+1}(\mathsf{ua}^{-1}([\mathsf{add}](p))\ (\mathsf{inr}(*))))\right)|_A\right) \\
&= \mathsf{ua}\left(\left(\mathsf{ua}^{-1}([\mathsf{add}](p) \cdot \mathfrak{m}_{n+1}(\mathsf{incr}(\mathsf{ua}^{-1}(p))\ (\mathsf{inr}(*))))\right)|_A\right) && \text{by (2.129)} \\
&\equiv \mathsf{ua}\left(\left(\mathsf{ua}^{-1}([\mathsf{add}](p) \cdot \mathfrak{m}_{n+1}(\mathsf{inr}(*)))\right)|_A\right) \\
&\equiv \mathsf{ua}\left(\left(\mathsf{ua}^{-1}([\mathsf{add}](p) \cdot \mathsf{refl}_{[\![n+1]\!]})\right)|_A\right) \\
&= \mathsf{ua}\left(\left(\mathsf{incr}(\mathsf{ua}^{-1}(p))\right)|_A\right) && \text{by (2.129)} \\
&= \mathsf{ua}(\mathsf{ua}^{-1}(p)) = p. && \square
\end{aligned}$$

Lemma 5.79. *For every $n : \mathbb{N}$, $A : \mathcal{U}$ and $p : A + \mathbf{1} = [\![n+1]\!]$, there is a 2-path*

$$[\mathsf{add}](\overline{p}) = p \cdot \mathfrak{m}_{n+1}(\overset{\bullet}{p}).$$

Proof. The claim is proved by the following chain of identities:

$$\begin{aligned}
[\mathsf{add}](\overline{p}) &\equiv [\mathsf{add}]\left(\mathsf{ua}\left(\left(\mathsf{ua}^{-1}(p \cdot \mathfrak{m}_{n+1}(\overset{\bullet}{p}))\right)|_A\right)\right) \\
&= \mathsf{ua}\left(\mathsf{incr}\left(\left(\mathsf{ua}^{-1}(p \cdot \mathfrak{m}_{n+1}(\overset{\bullet}{p}))\right)|_A\right)\right) && \text{by (2.128)} \\
&= \mathsf{ua}(\mathsf{ua}^{-1}(p \cdot \mathfrak{m}_{n+1}(\overset{\bullet}{p}))) && \text{by Corollary 2.124} \\
&= p \cdot \mathfrak{m}_{n+1}(\overset{\bullet}{p}). && \square
\end{aligned}$$

In order to relate to $\mathsf{del}_\bullet$ the combinatorial machinery so far described, we will use the following definition.

Definition 5.80. A function $\widetilde{\mathfrak{m}} : \Pi\,(n : \mathbb{N})\,.\,[\![n]\!] \to (\mathsf{pt}_n = \mathsf{pt}_n)$ is defined by induction:

$$\begin{aligned}
\widetilde{\mathfrak{m}}_{n+1}(\mathsf{inr}(*)) &:\equiv \mathsf{refl}_{\mathsf{pt}_{n+1}}, \\
\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(\mathsf{inr}(*))) &:\equiv \mathsf{tw}_{\mathsf{pt}_n}, \\
\widetilde{\mathfrak{m}}_{n+3}(\mathsf{inl}(\mathsf{inl}(i))) &:\equiv \mathsf{tw}_{\mathsf{pt}_{n+1}} \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+1}}
\end{aligned} \tag{5.81}$$

for every $i : [\![n+1]\!]$; all the other cases are obtained *ex falso*.

Lemma 5.82. *For every $n : \mathbb{N}$ and $i : [\![n]\!]$, there is a 2-path $[\mathfrak{f}^{\mathsf{b}}_n](\widetilde{\mathfrak{m}}_n(i)) = \mathfrak{m}_n(i)$.*

Proof. The claim follows by the computation rules of $\mathfrak{f}^{\mathsf{b}}$. By induction on n and i, we have:

- for every $n : \mathbb{N}$,

$$[\mathfrak{f}^{\mathsf{b}}_{n+1}](\widetilde{\mathfrak{m}}_{n+1}(\mathsf{inr}(*))) \equiv [\mathfrak{f}^{\mathsf{b}}_{n+1}](\mathsf{refl}_{\mathsf{pt}_{n+1}}) \equiv \mathsf{refl}_{[\![n+1]\!]} \equiv \mathfrak{m}_{n+1}(\mathsf{inr}(*));$$

- for every $n : \mathbb{N}$, using (5.42), we have

$$[\mathfrak{f}^{\mathsf{b}}_{n+2}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(\mathsf{inr}(*)))) \equiv [\mathfrak{f}^{\mathsf{b}}_{n+2}](\mathsf{tw}_{\mathsf{pt}_n}) = \gamma_{[\![n]\!]} \equiv \mathfrak{m}_{n+2}(\mathsf{inl}(\mathsf{inr}(*)));$$

- for every $n : \mathbb{N}$, assuming as inductive hypothesis a 2-path

$$[\mathfrak{f}^{\mathsf{b}}_{n+2}](\widetilde{\mathfrak{m}}_{n+2}(j)) = \mathfrak{m}_{n+2}(j) \quad \text{for any } j : [\![n+2]\!], \tag{5.83}$$

  we have, for every $i : [\![n+1]\!]$,

$$\begin{aligned}
& [\mathfrak{f}^{\mathsf{b}}_{n+3}](\widetilde{\mathfrak{m}}_{n+3}(\mathsf{inl}(\mathsf{inl}(i)))) \\
\equiv\ & [\mathfrak{f}^{\mathsf{b}}_{n+3}](\mathsf{tw}_{\mathsf{pt}_{n+1}} \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+1}}) \\
=\ & [\mathfrak{f}^{\mathsf{b}}_{n+3}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot [\mathfrak{f}^{\mathsf{b}}_{n+3}]([\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i)))) \cdot [\mathfrak{f}^{\mathsf{b}}_{n+3}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \\
=\ & \gamma_{[\![n+1]\!]} \cdot [\mathfrak{f}^{\mathsf{b}}_{n+3}]([\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i)))) \cdot \gamma_{[\![n+1]\!]} && \text{by (5.42)} \\
=\ & \gamma_{[\![n+1]\!]} \cdot [\mathsf{add}]([\mathfrak{f}^{\mathsf{b}}_{n+2}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i)))) \cdot \gamma_{[\![n+1]\!]} && \text{by (5.44)} \\
=\ & \gamma_{[\![n+1]\!]} \cdot [\mathsf{add}](\mathfrak{m}_{n+2}(\mathsf{inl}(i))) \cdot \gamma_{[\![n+1]\!]} && \text{by (5.83)} \\
\equiv\ & \mathfrak{m}_{n+3}(\mathsf{inl}(\mathsf{inl}(i))).
\end{aligned}$$

All other cases are obtained *ex falso*. □

The family of functions $\widetilde{\mathfrak{m}}$ satisfies the following lemma, which we will use later in this section.

Lemma 5.84. *For every $n : \mathbb{N}$ and $i : [\![n]\!]$, there is a 2-path*

$$\begin{aligned}
\widetilde{\mathsf{br}}_{n,i} :\ & \mathsf{tw}_{\mathsf{pt}_n} \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+1}(\mathsf{inl}(i))) \cdot \mathsf{tw}_{\mathsf{pt}_n} \\
& = [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+1}(\mathsf{inl}(i))) \cdot \mathsf{tw}_{\mathsf{pt}_n} \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+1}(\mathsf{inl}(i))).
\end{aligned} \tag{5.85}$$

Proof. By induction on n and i.

- for every $n : \mathbb{N}$, the term $\widetilde{\mathsf{br}}_{n+1,\mathsf{inr}(*)}$ is obtained directly by virtue of the constructor $\mathsf{br}_{\mathsf{pt}_n}$ of $\mathsf{del}_{(-)}$, since $\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(\mathsf{inr}(*))) :\equiv \mathsf{tw}_{\mathsf{pt}_n}$;

- for every $n : \mathbb{N}$, $i : [\![n+1]\!]$ and assuming given the term $\widetilde{\mathsf{br}}_{n+1,i}$ as inductive hypothesis, the term $\widetilde{\mathsf{br}}_{n+2,\mathsf{inl}(i)}$ is obtained as follows:

$$
\begin{aligned}
&\mathsf{tw}_{\mathsf{pt}_{n+2}} \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+3}(\mathsf{inl}(\mathsf{inl}(i)))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \\
&= \mathsf{tw}_{\mathsf{pt}_{n+2}} \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot [\mathsf{i}]([\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i)))) \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \\
&= \mathsf{tw}_{\mathsf{pt}_{n+2}} \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \\
&\qquad \cdot [\mathsf{i}]([\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i)))) \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} && \text{by } \mathsf{do}_{\mathsf{pt}_{n+2}} \\
&= \mathsf{tw}_{\mathsf{pt}_{n+2}} \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \\
&\qquad \cdot [\mathsf{i}]([\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i)))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} && \text{by (5.22)} \\
&= [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \\
&\qquad \cdot [\mathsf{i}]([\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i)))) \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) && \text{by } \mathsf{br}_{\mathsf{pt}_{n+1}} \\
&= [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \cdot [\mathsf{i}]([\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i)))) \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \\
&\qquad \cdot [\mathsf{i}]([\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i)))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) && \text{by } \widetilde{\mathsf{br}}_{n+1,i} \\
&= [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot [\mathsf{i}]([\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i)))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \\
&\qquad \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \cdot [\mathsf{i}]([\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i)))) \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) && \text{by (5.22)} \\
&= [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot [\mathsf{i}]([\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i)))) \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \\
&\qquad \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) \cdot [\mathsf{i}]([\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(i)))) \cdot [\mathsf{i}](\mathsf{tw}_{\mathsf{pt}_{n+1}}) && \text{by } \mathsf{br}_{\mathsf{pt}_{n+1}} \\
&= [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+3}(\mathsf{inl}(\mathsf{inl}(i)))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+2}} \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+3}(\mathsf{inl}(\mathsf{inl}(i)))).
\end{aligned}
$$

All other cases are obtained *ex falso*. □

With the lemmata shown so far, we can now prove the inductive case in Theorem 5.69 relative to the constructor i.

Proof of Lemma 5.64. Given $n : \mathbb{N}$, $a : \mathsf{del}_n$, terms $\mathfrak{e}^{\mathsf{b}}_{n,a}$ and $\mathfrak{e}^{\mathsf{f}}_{n,a}$ as in (5.65) and (5.66), and a path $p : \mathfrak{f}^{\mathsf{b}}_{n+1}(\mathsf{i}(a)) = [\![n+1]\!]$, we define

$$\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathfrak{e}^{\mathsf{b}}_{n,a}, p) :\equiv [\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n,a}(\overline{p})) \cdot \widetilde{\mathfrak{m}}_{n+1}(\overset{\bullet}{p}). \tag{5.86}$$

Then, the 2-path $\mathfrak{e}^{\mathsf{f}}_{n+1,\mathsf{i}(a)}(\mathfrak{e}^{\mathsf{b}}_{n,a}, \mathfrak{e}^{\mathsf{f}}_{n,a}, p) : [\mathfrak{f}^{\mathsf{b}}_{n+1}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathfrak{e}^{\mathsf{b}}_{n,a}, p)) = p$ is given by the following chain of identities:

$$
\begin{aligned}
&[\mathfrak{f}^{\mathsf{b}}_{n+1}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(a)}(\mathfrak{e}^{\mathsf{b}}_{n,a}, p)) \\
&\qquad \equiv [\mathfrak{f}^{\mathsf{b}}_{n+1}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n,a}(\overline{p})) \cdot \widetilde{\mathfrak{m}}_{n+1}(\overset{\bullet}{p})) \\
&\qquad = [\mathfrak{f}^{\mathsf{b}}_{n+1}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n,a}(\overline{p}))) \cdot [\mathfrak{f}^{\mathsf{b}}_{n+1}](\widetilde{\mathfrak{m}}_{n+1}(\overset{\bullet}{p})) \\
&\qquad = [\mathsf{add}]([\mathfrak{f}^{\mathsf{b}}_{n}](\mathfrak{e}^{\mathsf{b}}_{n,a}(\overline{p}))) \cdot [\mathfrak{f}^{\mathsf{b}}_{n+1}](\widetilde{\mathfrak{m}}_{n+1}(\overset{\bullet}{p})) && \text{by (5.44)}
\end{aligned}
$$

$$
\begin{aligned}
&= [\mathsf{add}](\overline{p}) \cdot [\mathfrak{f}^{\mathsf{b}}_{n+1}](\widetilde{\mathfrak{m}}_{n+1}(\overset{\bullet}{p})) && \text{by } \mathfrak{e}^{\mathsf{f}}_{n,a}(\overline{p}) \\
&= p \cdot \mathfrak{m}_{n+1}(\overset{\bullet}{p}) \cdot [\mathfrak{f}^{\mathsf{b}}_{n+1}](\widetilde{\mathfrak{m}}_{n+1}(\overset{\bullet}{p})) && \text{by Lemma 5.79} \\
&= p \cdot (\mathfrak{m}_{n+1}(\overset{\bullet}{p}))^{-1} \cdot [\mathfrak{f}^{\mathsf{b}}_{n+1}](\widetilde{\mathfrak{m}}_{n+1}(\overset{\bullet}{p})) && \text{by Lemma 5.72} \\
&= p \cdot ([\mathfrak{f}^{\mathsf{b}}_{n+1}](\widetilde{\mathfrak{m}}_{n+1}(\overset{\bullet}{p})))^{-1} \cdot [\mathfrak{f}^{\mathsf{b}}_{n+1}](\widetilde{\mathfrak{m}}_{n+1}(\overset{\bullet}{p})) && \text{by Lemma 5.82} \\
&= p. && \square
\end{aligned}
$$

For the proof of Lemma 5.67, we will need to fully harness the combinatorial structure of the subuniverse of finite types. We are interested, in particular, in determining $\overset{\bullet}{\overline{p}} : [\![n+1]\!]$ for $p : A + \mathbf{1} + \mathbf{1} = [\![n+2]\!]$; this will depend on both $\overset{\bullet}{p}$ and $\overset{\bullet\bullet}{p}$. We discussed $\mathsf{ua}^{-1}(\mathfrak{m}_{n+1}(i))\ (i)$ for $i : [\![n+1]\!]$ in Lemma 5.73; we will now provide a calculation of $\mathsf{ua}^{-1}(\mathfrak{m}_{n+2}(i))\ (j)$ when i and j are *different* terms.

Definition 5.87. Let $n : \mathbb{N}$, $i, j : [\![n+2]\!]$ and assume $d : (i = j) \to \mathbf{0}$. A term $\mathsf{w}_n(i, j, d) : [\![n+1]\!]$ is defined by induction on n, i and j, as follows:

$$
\begin{aligned}
&\mathsf{w}_0(\ i\ ,\ j\ , d) :\equiv \mathsf{inr}(*) && \text{for } i, j : [\![2]\!] \\
&\mathsf{w}_{n+1}(\ \mathsf{inr}(*)\ ,\ \mathsf{inr}(*)\ , d) && \text{is given } \textit{ex falso}, \text{ with } d(\mathsf{refl}) : \mathbf{0} \\
&\mathsf{w}_{n+1}(\ \mathsf{inr}(*)\ ,\ \mathsf{inl}(j)\ , d) :\equiv j && \text{for } j : [\![n+2]\!] \\
&\mathsf{w}_{n+1}(\mathsf{inl}(\mathsf{inr}(*))\ ,\ \mathsf{inr}(*)\ , d) :\equiv \mathsf{inr}(*) && \\
&\mathsf{w}_{n+1}(\mathsf{inl}(\mathsf{inr}(*))\ , \mathsf{inl}(\mathsf{inr}(*))\ , d) && \text{is given } \textit{ex falso}, \text{ with } d(\mathsf{refl}) : \mathbf{0} \\
&\mathsf{w}_{n+1}(\mathsf{inl}(\mathsf{inr}(*))\ ,\ \mathsf{inl}(\mathsf{inl}(j))\ , d) :\equiv \mathsf{inl}(j) && \text{for } j : [\![n+1]\!] \\
&\mathsf{w}_{n+1}(\ \mathsf{inl}(\mathsf{inl}(i))\ ,\ \mathsf{inr}(*)\ , d) :\equiv \mathsf{inl}(i) && \text{for } i : [\![n+1]\!] \\
&\mathsf{w}_{n+1}(\ \mathsf{inl}(\mathsf{inl}(i))\ , \mathsf{inl}(\mathsf{inr}(*))\ , d) :\equiv \mathsf{inr}(*) && \text{for } i : [\![n+1]\!] \\
&\mathsf{w}_{n+1}(\ \mathsf{inl}(\mathsf{inl}(i))\ ,\ \mathsf{inl}(\mathsf{inl}(j))\ , d) :\equiv \mathsf{inl}\,(\mathsf{w}_n(\mathsf{inl}(i), \mathsf{inl}(j),\ (p \mapsto d([\mathsf{inl}](p))))) \\
& && \text{for } i, j : [\![n+1]\!].
\end{aligned}
$$

We will omit the argument d from the notation.

The definition above is used in the following lemma, which determines, for every $i : [\![n+2]\!]$, where the symmetry $\mathsf{ua}^{-1}(\mathfrak{m}_{n+2}(i)) : [\![n+2]\!] \simeq [\![n+2]\!]$ "moves" a term j different from i.

Lemma 5.88. *For every $n : \mathbb{N}$, $i, j : [\![n+2]\!]$ and for $d : (i = j) \to \mathbf{0}$, there is a path*

$$\mathsf{ua}^{-1}(\mathfrak{m}_{n+2}(i))\ (j) =_{[\![n+2]\!]} \mathsf{inl}(\mathsf{w}_n(i, j)). \tag{5.89}$$

Proof. The proof proceeds by induction on n, i and j. Most of the cases are trivial; we present here the recursive case.

Assuming as inductive hypothesis a path as in (5.89) for some $n : \mathbb{N}$ and for all arguments i, j and d, we have, for $i, j : [\![n+1]\!]$ and $d : (\mathsf{inl}(\mathsf{inl}(i)) = \mathsf{inl}(\mathsf{inl}(j))) \to \mathbf{0}$,

$$\begin{aligned}
&\mathsf{ua}^{-1}(\mathfrak{m}_{n+3}(\mathsf{inl}(\mathsf{inl}(i)))) \ (\mathsf{inl}(\mathsf{inl}(j))) \\
&\quad \equiv \mathsf{ua}^{-1}(\gamma_{[\![n+1]\!]} \cdot [\mathsf{add}](\mathfrak{m}_{n+2}(\mathsf{inl}(i))) \cdot \gamma_{[\![n+1]\!]}) \ (\mathsf{inl}(\mathsf{inl}(j))) \\
&\quad = \omega_{[\![n+1]\!]}(\mathsf{incr}(\mathsf{ua}^{-1}(\mathfrak{m}_{n+2}(\mathsf{inl}(i)))) \ (\omega_{[\![n+1]\!]}(\mathsf{inl}(\mathsf{inl}(i))))) \\
&\quad \equiv \omega_{[\![n+1]\!]}(\mathsf{incr}(\mathsf{ua}^{-1}(\mathfrak{m}_{n+2}(\mathsf{inl}(i)))) \ (\mathsf{inl}(\mathsf{inl}(j)))) \\
&\quad \equiv \omega_{[\![n+1]\!]}(\mathsf{inl}(\mathsf{ua}^{-1}(\mathfrak{m}_{n+2}(\mathsf{inl}(i))) \ (\mathsf{inl}(j)))) \\
&\quad = \omega_{[\![n+1]\!]}(\mathsf{inl}(\mathsf{inl}(\mathsf{w}_n(\mathsf{inl}(i), \mathsf{inl}(j))))) \qquad \text{inductively} \\
&\quad \equiv \mathsf{inl}(\mathsf{inl}(\mathsf{w}_n(\mathsf{inl}(i), \mathsf{inl}(j)))) \\
&\quad \equiv \mathsf{inl}(\mathsf{w}_{n+1}(\mathsf{inl}(\mathsf{inl}(i)), \mathsf{inl}(\mathsf{inl}(j)))),
\end{aligned}$$

proving the case. □

Corollary 5.90. *For every $n : \mathbb{N}$, $a : \mathsf{del}_n$ and $p : \mathfrak{f}^{\mathsf{b}}_{n+2}(\mathsf{i}(\mathsf{i}(a))) = [\![n+2]\!]$, there are paths $\overset{\bullet}{\overline{p}} = \mathsf{w}_n(\overset{\bullet}{p}, \overset{\bullet\bullet}{p})$ and $\overset{\bullet}{\overline{\gamma_{\mathfrak{f}^{\mathsf{b}}_n(a)} \cdot p}} = \mathsf{w}_n(\overset{\bullet\bullet}{p}, \overset{\bullet}{p})$.*

Proof. We have:

$$\begin{aligned}
\mathsf{inl}(\overset{\bullet}{\overline{p}}) &=_{[\![n+2]\!]} \mathsf{inl}\left((\mathsf{ua}^{-1}(p \cdot \mathfrak{m}_{n+2}(\overset{\bullet}{p})))|_{\mathfrak{f}^{\mathsf{b}}_{n+1}(\mathsf{i}(a))} \ (\mathsf{inr}(*))\right) \\
&\equiv \mathsf{ua}^{-1}(p \cdot \mathfrak{m}_{n+2}(\overset{\bullet}{p})) \ (\mathsf{inl}(\mathsf{inr}(*))) \\
&= \mathsf{ua}^{-1}(\mathfrak{m}_{n+2}(\overset{\bullet}{p})) \ (\overset{\bullet\bullet}{p}) \\
&= \mathsf{inl}(\mathsf{w}_n(\overset{\bullet}{p}, \overset{\bullet\bullet}{p})) \qquad \text{by Lemma 5.88,}
\end{aligned}$$

where $\mathsf{w}_n(\overset{\bullet}{p}, \overset{\bullet\bullet}{p})$ is well-defined, since from a path $\overset{\bullet}{p} = \overset{\bullet\bullet}{p}$ one can obtain a path

$$\mathsf{inr}(*) = (\mathsf{ua}^{-1}(p))^{-1}(\overset{\bullet}{p}) = (\mathsf{ua}^{-1}(p))^{-1}(\overset{\bullet\bullet}{p}) = \mathsf{inl}(\mathsf{inr}(*)),$$

and thus a term in $\mathbf{0}$ (Lemma 2.70). Hence, by Lemma 2.72, $\overset{\bullet}{\overline{p}} = \mathsf{w}_n(\overset{\bullet}{p}, \overset{\bullet\bullet}{p})$. By the same reasoning, we obtain $\overset{\bullet}{\overline{\gamma \cdot p}} = \mathsf{w}_n\left((\gamma \cdot p)^{\bullet}, (\gamma \cdot p)^{\bullet\bullet}\right) = \mathsf{w}_n(\overset{\bullet\bullet}{p}, \overset{\bullet}{p})$. □

As mentioned, we will assume the following result on the combinatorics of finite types, which deals with properties of permutations. Although we believe in its provability, exhibiting an exact proof term revealed itself to be a challenging task.

Assumption 5.91. *Let $n : \mathbb{N}$ and $p : [\![n+2]\!] = [\![n+2]\!]$. The following holds:*

(i) if $\overset{\bullet}{p} = \mathsf{inr}(*)$, then $\overline{p} = \overline{\gamma_{[\![n]\!]} \cdot p}$;

(ii) if $\overset{\bullet}{p} = \mathsf{inl}(\mathsf{inr}(*))$ and $\overset{\bullet\bullet}{p} = \mathsf{inl}(\mathsf{inl}(j))$ for some $j : [\![n]\!]$, then $\overline{\overline{p}} = \overline{\overline{\gamma_{[\![n]\!]} \cdot p}}$.

Moreover, assume that a 2*-path as in (5.68) can be found for every* $p : [\![n+3]\!] = [\![n+3]\!]$ *such that* $\dot{p} = \mathsf{inl}(i)$ *and* $\ddot{p} = \mathsf{inl}(j)$ *for some* $i, j : [\![n+2]\!]$*. Then:*

(iii) a 2*-path as in (5.68) can be found for every* $q : [\![n+4]\!] = [\![n+4]\!]$ *such that* $\dot{q} = \mathsf{inl}(\mathsf{inl}(i))$ *and* $\ddot{q} = \mathsf{inl}(\mathsf{inl}(j))$.

The mathematical content of Assumption 5.91 is the following one, which we will explain using a mathematical notation disconnected from HoTT. A loop $p : [\![n+2]\!] = [\![n+2]\!]$ corresponds to a permutation $\sigma : \{1, \dots, n+2\} \simeq \{1, \dots, n+2\}$; accordingly, the path $\gamma_{[\![n]\!]} \cdot p$ corresponds to the permutation $\sigma \circ (n+2,\ n+1)$, where we precompose by the transposition of $n+2$ and $n+1$. The reduction $\overline{p} : [\![n+1]\!] = [\![n+1]\!]$ corresponds to the restriction to $\{1, \dots, n+1\}$ of the permutation

$$(n+2,\ \sigma(n+2)) \circ \sigma,$$

obtained by composing with the transposition of $n+2$ and $\sigma(n+2)$; the restriction can be applied, since, trivially, $\big((n+2,\ \sigma(n+2)) \circ \sigma\big)(n+2) = n+2$. The inductive definition of $[\![n]\!]$ allows us to keep track of some elements: for instance, the terms $\mathsf{inr}(*)$ and $\mathsf{inl}(\mathsf{inr}(*)) : [\![n+2]\!]$ stand for the elements $n+2$ and $n+1$, respectively. The terms $\dot{p}$ and $\ddot{p} : [\![n+2]\!]$ then correspond, respectively, to the elements $\sigma(n+2)$ and $\sigma(n+1)$.

Claim (i) in Assumption 5.91 represents the statement: if $\sigma(n+2) = n+2$, then

$$\big((n+2,\ \sigma(n+2)) \circ \sigma\big)(i) \ = \ \big((n+2,\ \sigma(n+1)) \circ \sigma \circ (n+2,\ n+1)\big)(i) \tag{5.92}$$

for every $i \leq n+1$. Substituting with the hypothesis, (5.92) reduces to:

$$\sigma(i) \ = \ \big((\sigma(n+2),\ \sigma(n+1)) \circ \sigma \circ (n+2,\ n+1)\big)(i) \tag{5.93}$$

for every $i \leq n+1$. This can be easily verified by case analysis (separately for $i = n+1$ and $i \leq n$). Similarly, claim (ii) represents the statement: if $\sigma(n+2) = n+1$ and $\sigma(n+1) = j \leq n$, then

$$\begin{aligned} &\Big(\big(n+1,\ \big((n+2,\ \sigma(n+2))(\sigma(n+1))\big)\big) \circ (n+2,\ \sigma(n+2)) \circ \sigma\Big)(i) \qquad (5.94)\\ &= \Big(\big(n+1,\ \big((n+2,\ \sigma(n+1))(\sigma(n+2))\big)\big) \circ (n+2,\ \sigma(n+1)) \circ \sigma \circ (n+2,\ n+1)\Big)(i) \end{aligned}$$

for every $i \leq n$. Substituting with the hypotheses and simplifying in (5.94), and using that $i \leq n$, this reduces to showing that

$$\big((n+1,\ j) \circ (n+2,\ n+1) \circ \sigma\big)(i) \ = \ \big((n+2,\ j) \circ \sigma\big)(i) \tag{5.95}$$

for every $i \leq n$. This can be shown again by case analysis: if $\sigma(i) = n+2$, then both the LHS and the RHS in (5.95) compute to j; the case for $\sigma(i) = n+1 = \sigma(n+2)$ does

not apply because $i \leq n$; finally, if $\sigma(i) \leq n$, then we notice that it is not the case that $\sigma(i) = j = \sigma(n+1)$ because $i \leq n$, and both the LHS and the RHS in (5.95) reduce to $\sigma(i)$.

While the informal proofs of claims (i) and (ii) in Assumption 5.91 are manifestly nothing more than an elementary exercise in combinatorics, a formalization in our framework would actually be quite cumbersome and require a number of preliminary results and nested induction on coproduct types. Especially in (ii), the argument by induction on $\sigma(i)$ would need to be formalized as a term in a type of the form

$$\Pi\,(i : [\![n]\!])\,.\,\Pi\,(x : [\![n+2]\!])\,.\,\left((\mathsf{ua}^{-1}(p)\;(\mathsf{inl}(\mathsf{inl}(i))) = x) \to \ldots\right),$$

which would take considerably more effort than the simple proof above. To this, one should also remember that the definition of $\overline{\overline{p}}$ entails going back and forth with univalence *twice*, which increases the overall complexity of the sought proof term.

Claim (iii) is of a different kind. Plain and simple, it is the inductive step in the proof that the definition of $\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{i}(-)}$ from Lemma 5.64 (see Fig. 5.6) "respects" transpositions of adjacent elements, in the sense that, for every $n : \mathbb{N}$, the function

$$\mathfrak{e}^{\mathsf{b}}_{n+2,\mathsf{pt}_{n+2}} \;:\; ([\![n+2]\!] = [\![n+2]\!]) \to (\mathsf{pt}_{n+2} = \mathsf{pt}_{n+2})$$

should recursively take, for every $p : [\![n+2]\!] = [\![n+2]\!]$, the path $\gamma_{[\![n]\!]} \cdot p$ to the path $\mathsf{tw}_a \cdot \mathfrak{e}^{\mathsf{b}}_{n+2,\mathsf{pt}_{n+2}}(p)$. Since the "combinatorial" semantics of $\mathcal{BS}_\bullet$ and of $\mathsf{del}_\bullet$ is the same, we find no help in reasoning on the nature of the permutations involved in the definition of each subterm of the type in (5.68); currently, this problem remains unsolved.

Using Assumption 5.91, which regrettably makes our formalization incomplete, we are able to prove the last lemma in the way of an equivalence $\mathsf{del}_\bullet \simeq \mathcal{BS}_\bullet$.

Proof of Lemma 5.67. Let $n : \mathbb{N}$, $p : [\![n+2]\!] = [\![n+2]\!]$ and $\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}$ as in (5.65). Using the definition of $\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}$ given in the proof of Lemma 5.64 for $a :\equiv \mathsf{pt}_n$, to find a 2-path as in (5.68) is to find a 2-path

$$\begin{aligned} &[\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}(\overline{\overline{\gamma_{[\![n]\!]} \cdot p}}))) \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+1}(\overline{\overset{\bullet}{\gamma_{[\![n]\!]} \cdot p}})) \cdot \widetilde{\mathfrak{m}}_{n+2}((\gamma_{[\![n]\!]} \cdot p)^{\bullet}) \\ = \mathsf{tw}_{\mathsf{pt}_n} \cdot\; &[\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}(\overline{\overline{p}}))) \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+1}(\overset{\bullet}{\overline{p}})) \cdot \widetilde{\mathfrak{m}}_{n+2}(\overset{\bullet}{p}), \end{aligned} \tag{5.96}$$

or, via Lemma 5.21 and simplifying,

$$\begin{aligned} &[\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}(\overline{\overline{\gamma_{[\![n]\!]} \cdot p}}))) \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+1}(\overline{\overset{\bullet}{\gamma_{[\![n]\!]} \cdot p}})) \cdot \widetilde{\mathfrak{m}}_{n+2}(\overset{\bullet\bullet}{p}) \\ = \;&[\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}(\overline{\overline{p}}))) \cdot \mathsf{tw}_{\mathsf{pt}_n} \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+1}(\overset{\bullet}{\overline{p}})) \cdot \widetilde{\mathfrak{m}}_{n+2}(\overset{\bullet}{p}). \end{aligned} \tag{5.97}$$

We proceed by induction on $n : \mathbb{N}$.

- For $p : [\![2]\!] = [\![2]\!]$, we have that $\overline{\overline{\gamma_{\mathbf{0}} \cdot p}} = \overline{\overline{p}}$, as the type $\mathbf{0} = \mathbf{0}$ is a (-1)-type; moreover, $\overset{\bullet}{p} = \mathsf{w}_0(\overset{\bullet}{p}, \overset{\bullet\bullet}{p}) \equiv \mathsf{inr}(*)$ by Corollary 5.90, and similarly $\overset{\bullet}{\gamma_{\mathbf{0}} \cdot p} = \mathsf{inr}(*)$, so $\widetilde{\mathfrak{m}}_1(\overset{\bullet}{\gamma_{\mathbf{0}} \cdot p}) = \widetilde{\mathfrak{m}}_1(\overset{\bullet}{p}) = \widetilde{\mathfrak{m}}_1(\mathsf{inr}(*)) \equiv \mathsf{refl}_{\mathsf{pt}_1}$. Hence, it is enough to construct a 2-path $\widetilde{\mathfrak{m}}_2(\overset{\bullet\bullet}{p}) = \mathsf{tw}_{\mathsf{pt}_0} \cdot \widetilde{\mathfrak{m}}_2(\overset{\bullet}{p})$. Induction on $\overset{\bullet}{p}$ and $\overset{\bullet\bullet}{p} : [\![2]\!] \equiv [2]$ gives the following cases:
  - if $\overset{\bullet}{p} \equiv \mathsf{inr}(*)$ and $\overset{\bullet\bullet}{p} \equiv \mathsf{inl}(\mathsf{inr}(*))$, we obtain immediately
$$\widetilde{\mathfrak{m}}_2(\overset{\bullet\bullet}{p}) \equiv \mathsf{tw}_{\mathsf{pt}_0} = \mathsf{tw}_{\mathsf{pt}_0} \cdot \mathsf{refl}_{\mathsf{pt}_2} \equiv \mathsf{tw}_{\mathsf{pt}_0} \cdot \widetilde{\mathfrak{m}}_2(\overset{\bullet}{p});$$
  - if $\overset{\bullet}{p} \equiv \mathsf{inl}(\mathsf{inr}(*))$ and $\overset{\bullet\bullet}{p} \equiv \mathsf{inr}(*)$, we use $\mathsf{do}_{\mathsf{pt}_0}$ to get
$$\widetilde{\mathfrak{m}}_2(\overset{\bullet\bullet}{p}) \equiv \mathsf{refl}_{\mathsf{pt}_2} = \mathsf{tw}_{\mathsf{pt}_0} \cdot \mathsf{tw}_{\mathsf{pt}_0} \equiv \mathsf{tw}_{\mathsf{pt}_0} \cdot \widetilde{\mathfrak{m}}_2(\overset{\bullet}{p});$$

  all other cases are obtained *ex falso*, as assuming $\overset{\bullet}{p} = \overset{\bullet\bullet}{p}$ leads to a term in $\mathbf{0}$ (as in the proof of Corollary 5.90).
- Assuming $n : \mathbb{N}$ and (5.97) for all $q : [\![n+2]\!] = [\![n+2]\!]$, and given $p : [\![n+3]\!] = [\![n+3]\!]$, we reason again by induction on $\overset{\bullet}{p}$ and $\overset{\bullet\bullet}{p} : [\![n+3]\!]$, excluding the cases $\overset{\bullet}{p} = \overset{\bullet\bullet}{p}$, which are resolved *ex falso*:
  (i) if $\overset{\bullet}{p} \equiv \mathsf{inr}(*)$ and $\overset{\bullet\bullet}{p} \equiv \mathsf{inl}(\mathsf{inr}(*))$, we have that $\overset{\bullet}{\overline{p}} = \mathsf{w}_{n+1}(\overset{\bullet}{p}, \overset{\bullet\bullet}{p}) \equiv \mathsf{inr}(*)$ by Corollary 5.90. We find:
$$\begin{aligned}
&[\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{\gamma_{[\![n+1]\!]} \cdot p}}))) \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\overset{\bullet}{\gamma_{[\![n+1]\!]} \cdot p})) \cdot \widetilde{\mathfrak{m}}_{n+3}(\overset{\bullet\bullet}{p}) \\
&= [\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{p}}))) \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\overset{\bullet}{\overline{p}})) \cdot \widetilde{\mathfrak{m}}_{n+3}(\overset{\bullet\bullet}{p}) \qquad \text{by Ass. 5.91(i)} \\
&= [\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{p}}))) \cdot \mathsf{refl}_{\mathsf{pt}_{n+3}} \cdot \mathsf{tw}_{\mathsf{pt}_{n+1}} \\
&= [\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{p}}))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+1}} \cdot \mathsf{refl}_{\mathsf{pt}_{n+3}} \cdot \mathsf{refl}_{\mathsf{pt}_{n+3}} \\
&= [\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{p}}))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+1}} \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\overset{\bullet}{\overline{p}})) \cdot \widetilde{\mathfrak{m}}_{n+3}(\overset{\bullet}{p});
\end{aligned}$$
  (ii) if $\overset{\bullet}{p} \equiv \mathsf{inr}(*)$ and $\overset{\bullet\bullet}{p} \equiv \mathsf{inl}(\mathsf{inl}(j))$ for some $j : [\![n+1]\!]$, we have that $\overset{\bullet}{\overline{p}} = \mathsf{w}_{n+1}(\overset{\bullet}{p}, \overset{\bullet\bullet}{p}) = \mathsf{inl}(j)$ by Corollary 5.90. We find:
$$\begin{aligned}
&[\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{\gamma_{[\![n+1]\!]} \cdot p}}))) \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\overset{\bullet}{\gamma_{[\![n+1]\!]} \cdot p})) \cdot \widetilde{\mathfrak{m}}_{n+3}(\overset{\bullet\bullet}{p}) \\
&= [\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{p}}))) \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\overset{\bullet}{\overline{p}})) \cdot \widetilde{\mathfrak{m}}_{n+3}(\overset{\bullet\bullet}{p}) \qquad \text{by Ass. 5.91(i)} \\
&= [\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{p}}))) \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(j))) \\
&\qquad\qquad \cdot \mathsf{tw}_{\mathsf{pt}_{n+1}} \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(j))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+1}}
\end{aligned}$$

$$= [\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{p}}))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+1}} \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(j))) \quad \text{by Lemma 5.84}$$
$$= [\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{p}}))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+1}} \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\overset{\bullet}{p})) \cdot \widetilde{\mathfrak{m}}_{n+3}(\overset{\bullet}{p});$$

(iii) the case for $\overset{\bullet}{p} \equiv \mathsf{inl}(\mathsf{inr}(*))$ and $\overset{\bullet\bullet}{p} \equiv \mathsf{inr}(*)$ is analogous to the one presented in (i), with the roles of p and $\gamma_{[\![n+1]\!]} \cdot p$ exchanged;

(iv) if $\overset{\bullet}{p} \equiv \mathsf{inl}(\mathsf{inr}(*))$ and $\overset{\bullet\bullet}{p} \equiv \mathsf{inl}(\mathsf{inl}(j))$ for some $j : [\![n+1]\!]$, we have that $\overline{\overset{\bullet}{p}} = \mathsf{w}_{n+1}(\overset{\bullet}{p}, \overset{\bullet\bullet}{p}) = \mathsf{inl}(j)$ and $\overline{\overset{\bullet}{\gamma_{[\![n+1]\!]} \cdot p}} = \mathsf{w}_{n+1}(\overset{\bullet\bullet}{p}, \overset{\bullet}{p}) = \mathsf{inr}(*)$ by Corollary 5.90. We then find:

$$[\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{\gamma_{[\![n+1]\!]} \cdot p}}))) \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\overset{\bullet}{\gamma_{[\![n+1]\!]} \cdot p})) \cdot \widetilde{\mathfrak{m}}_{n+3}(\overset{\bullet\bullet}{p})$$
$$= [\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{p}}))) \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\overset{\bullet}{\gamma_{[\![n+1]\!]} \cdot p})) \cdot \widetilde{\mathfrak{m}}_{n+3}(\overset{\bullet\bullet}{p})$$
by Ass. 5.91(ii)
$$= [\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{p}}))) \cdot \mathsf{refl}_{\mathsf{pt}_{n+3}} \cdot \mathsf{tw}_{\mathsf{pt}_{n+1}} \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\mathsf{inl}(j))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+1}}$$
$$= [\mathsf{i}]([\mathsf{i}](\mathfrak{e}^{\mathsf{b}}_{n+1,\mathsf{pt}_{n+1}}(\mathfrak{e}^{\mathsf{b}}_{n,\mathsf{pt}_n}, \overline{\overline{p}}))) \cdot \mathsf{tw}_{\mathsf{pt}_{n+1}} \cdot [\mathsf{i}](\widetilde{\mathfrak{m}}_{n+2}(\overset{\bullet}{p})) \cdot \widetilde{\mathfrak{m}}_{n+3}(\overset{\bullet}{p});$$

(v) the case for $\overset{\bullet}{p} \equiv \mathsf{inl}(\mathsf{inl}(i))$ and $\overset{\bullet\bullet}{p} \equiv \mathsf{inr}(*)$ is analogous to the one in (ii);

(vi) the case for $\overset{\bullet}{p} \equiv \mathsf{inl}(\mathsf{inl}(i))$ and $\overset{\bullet\bullet}{p} \equiv \mathsf{inl}(\mathsf{inr}(*))$ is analogous to the one in (iv);

(vii) finally, assume that the 2-path can be found for every $p : [\![n+3]\!] = [\![n+3]\!]$ such that $\overset{\bullet}{p} = \mathsf{inl}(i)$ and $\overset{\bullet\bullet}{p} = \mathsf{inl}(j)$ for $i, j : [\![n+2]\!]$. Then, by Assumption 5.91(iii), we get the sought 2-path for $p : [\![n+4]\!] = [\![n+4]\!]$ such that $\overset{\bullet}{p} \equiv \mathsf{inl}(\mathsf{inl}(i))$ and $\overset{\bullet\bullet}{p} \equiv \mathsf{inl}(\mathsf{inl}(j))$, concluding the proof by induction. □

Remark 5.98. The proof of Theorem 5.69 we showed in this section can be alternatively presented as follows. As proved in [Uni13, Theorem 8.8.1], a function $f : A \to B$ is an equivalence if it is an embedding and the induced function $\|f\|_0 : \|A\|_0 \to \|B\|_0$ on 0-truncations is *surjective*. Combining this result with connectedness of the types in the family $\mathsf{del}_\bullet$ (Lemma 5.26), we then have that, for $n : \mathbb{N}$, the function $\mathfrak{f}_n : \mathsf{del}_n \to \mathcal{BS}_n$ is an equivalence if the following conditions hold:

(i) the function $\|\mathfrak{f}_n\|_0 : \|\mathsf{del}_n\|_0 \to \|\mathcal{BS}_n\|_0$ is surjective; this is trivial, as both del_n and $\mathcal{BS}_n$ are connected;

(ii) for every $a : \mathsf{del}_n$, the function

$$\|[\mathfrak{f}_n]\|_0 : \|a = \mathsf{pt}_n\|_0 \to \|\mathfrak{f}_n(a) = \mathfrak{f}_n(\mathsf{pt}_n)\|_0$$

is surjective;

(iii) for every $a : \mathsf{del}_n$ and $p, q : a = \mathsf{pt}_n$, the function

$$\|[[\mathfrak{f}_n]]\|_0 : \|p = q\|_0 \to \|[\mathfrak{f}_n](p) = [\mathfrak{f}_n](q)\|_0$$

is surjective; this is also trivial, since both del_n and $\mathcal{B}\mathsf{S}_n$ are 1-types;

(iv) for every $a : \mathsf{del}_n$, $p, q : a = \mathsf{pt}_n$ and $r, s : p = q$, the function

$$[[[\mathfrak{f}_n]]] : (r = s) \to ([[\mathfrak{f}_n]](r) = [[\mathfrak{f}_n]](s))$$

is an equivalence; as both the source and target of $[[[\mathfrak{f}_n]]]$ are (-1)-types, this simply entails defining a function in the opposite direction.

Trying to pursue such a proof of Theorem 5.69 would not reduce its complexity, which is largely due to the combinatorics of finite types. Roughly, condition (ii) corresponds to Lemma 5.63 and Lemma 5.64, while condition (iv) corresponds to Lemma 5.67.

5.5 Discussion and Conclusions

Save for the unformalized results in Assumption 5.91 about combinatorics of finite types, in this and the previous chapter we have managed to construct a chain of symmetric monoidal equivalences:

$$\mathsf{FSMG}(\mathbf{1}) \simeq \mathsf{slist}(\mathbf{1}) \simeq \mathsf{del}_\bullet \simeq \mathcal{B}\mathsf{S}_\bullet, \tag{5.99}$$

parallel to the one in (3.60) for free monoidal groupoids.

Especially in the proof of $\mathsf{del}_\bullet \simeq \mathcal{B}\mathsf{S}_\bullet$, univalence played a key role, because we essentially establish an equivalence between paths in $\mathsf{del}_\bullet$ and paths in the subuniverse of finite types. A path between finite types is much better described by the equivalence it entails; for example, a path $p : A + \mathbf{1} = A + \mathbf{1}$ conceals the information on where the equivalence $\mathsf{ua}^{-1}(p) : A + \mathbf{1} \simeq A + \mathbf{1}$ sends the rightmost term $\mathsf{inr}(*)$. Working with such an equivalence – rather than a path – lets us perform *induction* on the image of $\mathsf{inr}(*)$, which is necessary because the combinatorial nature of finite types (and of symmetric groups) is fundamentally an inductive machinery. Univalence allows us to switch between the notions of paths and equivalences, so that we can use the former to link identity types in $\mathsf{del}_\bullet$ to those in $\mathcal{B}\mathsf{S}_\bullet$ (via $[\mathfrak{f}_n]$) and the latter to reason about combinatorics.

The first, immediate conclusion we can draw from (5.99) is that the subuniverse of finite types is a *free* symmetric monoidal groupoid. Tracing the image of the generator $* : \mathbf{1}$ along the chain of equivalences described in this chapter, we see that

it corresponds to the (canonical) finite type $[1]$, as displayed in Fig. 5.8. In particular, the identity types in the subuniverse of finite types embody the symmetric monoidal structures of $\mathsf{FSMG}(\mathbf{1})$ and $\mathsf{slist}(\mathbf{1})$; that is, the constructors of FSMG, expressing associativity, unitality, symmetry and the coherence diagrams, describe the classifying spaces of symmetric groups. In turn, automorphisms of finite types can be described in terms of transpositions of adjacent elements in a symmetric list.

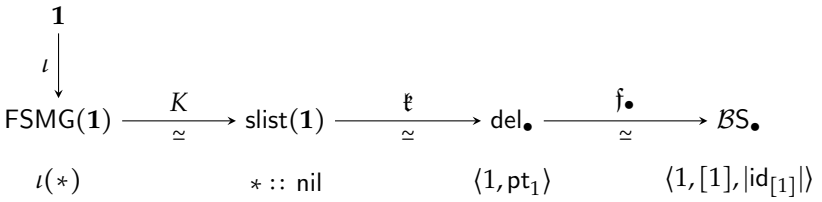


Figure 5.8: The generator $* : \mathbf{1}$ in $\mathsf{FSMG}(\mathbf{1})$ corresponds to the finite type $[1]$ in $\mathcal{B}\mathsf{S}_\bullet$.

The constructions $\mathsf{FSMG}(\mathbf{1})$, $\mathsf{slist}(\mathbf{1})$ and $\mathsf{del}_\bullet$ all provide elimination principles that can be employed, according to the need, to eliminate out of finite types. Indeed, assuming function extensionality, for every family $P : \mathcal{B}\mathsf{S}_\bullet \to \mathcal{U}$, we can construct a chain of equivalences between dependent function types:

$$\begin{aligned}\Pi\,(Z : \mathcal{B}\mathsf{S}_\bullet)\,.\,P(Z) &\simeq \Pi\,(a : \mathsf{del}_\bullet)\,.\,P(\mathfrak{f}_\bullet(a)) \\ &\simeq \Pi\,(l : \mathsf{slist}(\mathbf{1}))\,.\,P(\mathfrak{f}_\bullet(\mathfrak{t}(l))) \\ &\simeq \Pi\,(a : \mathsf{FSMG}(\mathbf{1}))\,.\,P(\mathfrak{f}_\bullet(\mathfrak{t}(K(a)))),\end{aligned} \tag{5.100}$$

where K, $\mathfrak{t}$ and $\mathfrak{f}_\bullet$ are the equivalences used in (5.99), which we described in detail in this thesis.

In this chapter we have so far focussed on symmetric monoidal groupoids generated by one element only (the term in $\mathbf{1}$). As a matter of fact, the chains of equivalences in (5.99) and (5.100) can be extended to a stronger result and encompass symmetric monoidal groupoids generated by any set, as presented in the following lemma.

Lemma 5.101. *Let X be a 0-type. There is a symmetric monoidal equivalence*

$$\mathsf{slist}(X) \simeq \left(\Sigma\,(a : \mathsf{del}_\bullet)\,.\,(\mathfrak{f}_\bullet^{\mathsf{b}}(a) \to X)\right).$$

Proof. The proof of Theorem 5.35 can be adapted to produce the sought symmetric monoidal equivalence:

- the type $\Sigma\,(a : \mathsf{del}_\bullet)\,.\,(\mathfrak{f}^{\mathsf{b}}_\bullet(a) \to X)$ has a symmetric monoidal structure obtained combining the one of $\mathsf{del}_\bullet$ (Lemma 5.30) with one on $\Sigma\,(A : \mathcal{U})\,.\,A \to X$; the latter uses $\langle \mathbf{0}, \mathsf{rec}_{\mathbf{0}} \rangle$ as unit, while the product of $\langle A, f \rangle$ and $\langle B, g \rangle$ is $\langle A + B, f + g \rangle$;
- a function $\widehat{\mathfrak{k}} : \mathsf{slist}(X) \to \left(\Sigma\,(a : \mathsf{del}_\bullet)\,.\,(\mathfrak{f}^{\mathsf{b}}_\bullet(a) \to X)\right)$ can be defined so that

$$\widehat{\mathfrak{k}}(\mathsf{nil}) :\equiv \langle \mathfrak{k}(\mathsf{nil}), \mathsf{rec}_{\mathbf{0}} \rangle \equiv \langle \underline{\mathsf{pt}_0}, \mathsf{rec}_{\mathbf{0}} \rangle,$$
$$\widehat{\mathfrak{k}}(x :: l) :\equiv \langle \mathfrak{k}(x :: l), \mathsf{pr}_2(\widehat{\mathfrak{k}}(l)) + \mathsf{const}_x \rangle,$$

  for every $x : X$ and $l : \mathsf{slist}(X)$, with $\mathsf{const}_x :\equiv (u \mapsto x) : \mathbf{1} \to X$ and $\mathfrak{k}$ as in Definition 5.32. Indeed, we have that

$$\mathfrak{f}^{\mathsf{b}}_\bullet(\mathfrak{k}(x :: l)) \equiv \mathfrak{f}^{\mathsf{b}}_\bullet\langle \mathsf{pr}_1(\mathfrak{k}(l)) + 1,\ \mathsf{i}(\mathsf{pr}_2(\mathfrak{k}(l)))\rangle \equiv \mathfrak{f}^{\mathsf{b}}_\bullet(\mathfrak{k}(l)) + \mathbf{1},$$

  so $\mathsf{pr}_2(\widehat{\mathfrak{k}}(l)) + \mathsf{const}_x : \mathfrak{f}^{\mathsf{b}}_\bullet(\mathfrak{k}(x :: l)) \to X$ is well-typed. The requirements relative to the higher constructors can be given in a way similar to the one presented in Definition 5.32; the function $\widehat{\mathfrak{k}}$ can then be proved to be the underlying function of a symmetric monoidal functor;
- conversely, a function $\widehat{\mathfrak{j}} : \left(\Sigma\,(a : \mathsf{del}_\bullet)\,.\,(\mathfrak{f}^{\mathsf{b}}_\bullet(a) \to X)\right) \to \mathsf{slist}(X)$ is produced similarly to $\mathfrak{j}$ in Definition 5.34, i.e., via a term

$$j : \Pi\,(n : \mathbb{N})\,.\,\Pi\,(a : \mathsf{del}_n)\,.\,(\mathfrak{f}^{\mathsf{b}}_n(a) \to X) \to \mathsf{slist}(X)$$

  obtained by the elimination principle of $\mathsf{del}_{(-)}$. The resulting dependent function will compute

$$j(0, \mathsf{pt}_0, f) :\equiv \mathsf{nil},$$
$$j(n + 1, \mathsf{i}(a), f) :\equiv f(\mathsf{inr}(*)) :: j(n, a, f \circ \mathsf{inl})$$

  for every $n : \mathbb{N}$, $a : \mathsf{del}_n$ and $f : \mathfrak{f}^{\mathsf{b}}_n(a) \to X$, where $\mathfrak{f}^{\mathsf{b}}_{n+1}(\mathsf{i}(a)) \equiv \mathfrak{f}^{\mathsf{b}}_n(a) + \mathbf{1}$, thus $f(\mathsf{inr}(*)) : X$ and $f \circ \mathsf{inl} : \mathfrak{f}^{\mathsf{b}}_n(a) \to X$ are well-typed. Again, the requirements relative to the higher constructors can be given similarly to Definition 5.34;
- the equivalence is proved in the same way as in Theorem 5.35. □

The lemma above has not been formalized, as the path algebra required to formally prove it is rather involved.

Remark 5.102. For a 0-type X, the type $\Sigma\,(Z : \mathcal{BS}_\bullet)\,.\,(Z \to X)$ – where, to simplify the notation, we liberally write Z instead of its type component $\mathsf{pr}_1(\mathsf{pr}_2(Z)) : \mathcal{U}$ – also has a symmetric monoidal structure, which combines the one already known for $\mathcal{BS}_\bullet$ (Lemma 5.11) with the on $\Sigma\,(A : \mathcal{U})\,.\,A \to X$ described in Lemma 5.101.

By virtue of the previous lemma and the results presented in Chapter 4, we can conclude that the type of free symmetric monoidal expressions on X, defined inductively as the HIT $\mathsf{FSMG}(X)$, is equivalent to the type of unordered (*coordinate-free*), finite "vectors" with entries in the 0-type X.

Corollary 5.103. *For every 0-type X, there is a symmetric monoidal equivalence*

$$\mathsf{FSMG}(X) \simeq \Sigma\,(Z : \mathcal{B}\mathsf{S}_\bullet)\,.\,(Z \to X).$$

Proof. The claim is proved by the following chain of equivalences:

$$\begin{aligned}
\mathsf{FSMG}(X) &\simeq \mathsf{slist}(X) && \text{by Corollary 4.49}\\
&\simeq \Sigma\,(a : \mathsf{del}_\bullet)\,.\,(\mathfrak{f}^{\mathsf{b}}_\bullet(a) \to X) && \text{by Lemma 5.101}\\
&\simeq \Sigma\,(Z : \mathcal{B}\mathsf{S}_\bullet)\,.\,(Z \to X),
\end{aligned}$$

where the last equivalence is given by $\langle \mathfrak{f}_\bullet, (a \mapsto \mathsf{id})\rangle$. □

We can put this equivalence to use in the following way. In Chapter 4, we gave an example of a non-commuting diagram in $\mathsf{FSMG}(X)$ (Fig. 4.1), involving a symmetric monoidal expression where certain elements were repeated. Indeed, one can show that $(\tau_{a,a} = \mathsf{refl}_{a\otimes a}) \to \mathbf{0}$ for every $a : \mathsf{FSMG}(X)$, corresponding to the non-identification between the automorphisms ω_A and $\mathsf{id}_{A+\mathbf{1}+\mathbf{1}}$, for any $A : \mathcal{U}$. This, however, is the only pathological case; all diagrams in $\mathsf{FSMG}(X)$ involving expressions containing terms in X "different" from each other commute. Trying to classify such expressions in $\mathsf{FSMG}(X)$ would be a complex task; instead, the equivalence in Corollary 5.103 gives us an easy way to describe them as specific coordinate-free, finite vectors with distinct entries in X, i.e., to terms inhabiting the subtype of $\Sigma\,(Z : \mathcal{B}\mathsf{S}_\bullet)\,.\,(Z \to X)$ consisting of functions which are *embeddings*.

This result will be proved in the last theorem of this chapter. We consider it an alternative take on the coherence theorem for symmetric monoidal groupoids, as it complements the now established description of the paths in $\mathsf{FSMG}(X)$ (in terms of automorphisms of finite types) with a characterization of the class of commuting diagrams in the groupoid:

- all diagrams not containing instances of τ commute. Indeed, there is an obvious map $\mathsf{incl} : \mathsf{FMG}(X) \to \mathsf{FSMG}(X)$ obtained by the elimination principle of FMG, whose computation rules show that $\mathsf{incl} \circ \iota \equiv \iota$, i.e., the diagram in Fig. 5.9 commutes judgmentally;

- all (other) diagrams commute if and only if the corresponding diagrams in $\mathcal{B}\mathsf{S}_\bullet$ do; in particular, all diagrams involving symmetric monoidal expressions with distinct elements commute.

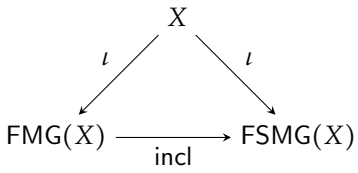


Figure 5.9: Inclusion of $\mathsf{FMG}(X)$ in $\mathsf{FSMG}(X)$.

The proof of the following theorem does not require any of the results previously shown in this chapter. However, it gains relevance in light of the equivalence in Corollary 5.103, which shows that, for a 0-type X, the symmetric monoidal groupoid $\Sigma\,(Z : \mathcal{B}\mathsf{S}_\bullet)\,.\,(Z \to X)$ is *free*.

Theorem 5.104 (Coherence for coordinate-free, finite vectors)**.** *Let X be a* 0*-type. The type*

$$\Sigma\,(Z : \mathcal{B}\mathsf{S}_\bullet)\,.\,\Sigma\,(f : Z \to X)\,.\,\mathsf{IsEmb}(f)$$

of coordinate-free vectors with distinct *entries in X is a* 0*-type.*

Proof. The statement holds in more generality: for any type X, the type

$$\Sigma\,(Z : \mathcal{U})\,.\,\Sigma\,(f : Z \to X)\,.\,\mathsf{IsEmb}(f)$$

is a 0-type. In order to prove the claim, we need to show that, for every $A, B : \mathcal{U}$, $f : A \to X$, $g : B \to X$, $h : \mathsf{IsEmb}(f)$ and $k : \mathsf{IsEmb}(g)$, the type

$$\langle A, f, h\rangle = \langle B, g, k\rangle$$

is a (-1)-type. Since being an embedding is a (-1)-type (Lemma 2.120(i)), it is enough to show that the type

$$\langle A, f\rangle =_{(\Sigma(Z:\mathcal{U}).\,(Z\to X))} \langle B, g\rangle \tag{5.105}$$

is a (-1)-type. By Remark 2.68, the type in (5.105) is equivalent to the type

$$\Sigma\,(p : A = B)\,.\,\left(p_*^{(Z\mapsto(Z\to X))}(f) = g\right). \tag{5.106}$$

In turn, the type in (5.106) is equivalent, using Remark 2.99, to the type

$$\Sigma\,(e : A \simeq B)\,.\,(f \sim g \circ e), \tag{5.107}$$

via the equivalence $\mathsf{ua}^{-1} : (A = B) \simeq (A \simeq B)$ on the base, and a family of equivalences

$$\prod (p : A = B) . \left(\left(p_*^{(Z \mapsto (Z \to X))}(f) = g \right) \simeq \left(f \simeq g \circ \mathsf{ua}^{-1}(p) \right) \right)$$

obtained by induction on p, which amounts to providing the equivalence $\mathsf{fxt}^{-1} : (f = g) \simeq (f \sim g)$. Finally, the type in (5.107) is equivalent to

$$\prod (x : X) . \mathsf{fib}_f(x) \simeq \mathsf{fib}_g(x). \tag{5.108}$$

Briefly explained, a family of equivalences s in the type in (5.108) determines an equivalence $\langle \mathsf{id}, s \rangle : \sum (x : X) . \mathsf{fib}_f(x) \simeq \sum (x : X) . \mathsf{fib}_g(x)$, and hence an equivalence

$$e : A \simeq \left(\sum (x : X) . \mathsf{fib}_f(x) \right) \simeq \left(\sum (x : X) . \mathsf{fib}_g(x) \right) \simeq B$$

by [Uni13, Lemma 4.82], such that $f \sim g \circ e$. Conversely, an equivalence $e : A \simeq B$ and a homotopy $f \sim g \circ e$ determine, for every $x : X$, an equivalence

$$\mathsf{fib}_f(x) \simeq \mathsf{fib}_{g \circ e}(x) \simeq \mathsf{fib}_g(x)$$

as one can show that fibers respect homotopies and equivalences. This correspondence is an equivalence; details are shown in [Gyl20, Lemma 5].

By univalence, the type in (5.108) is equivalent to

$$\prod (x : X) . \mathsf{fib}_f(x) = \mathsf{fib}_g(x),$$

which is a (-1)-type if, for every $x : X$, $\mathsf{fib}_g(x)$ is a (-1)-type (Lemma 2.130). This holds by Lemma 2.110, as g is an embedding. □

5.6 Figures in Proofs

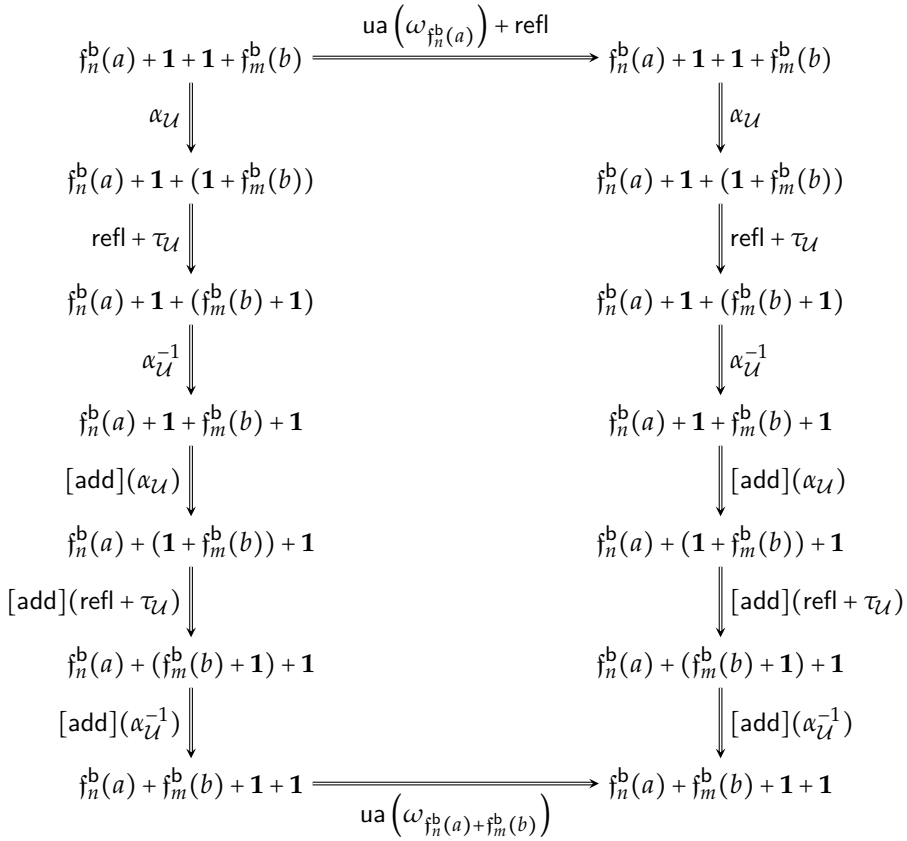
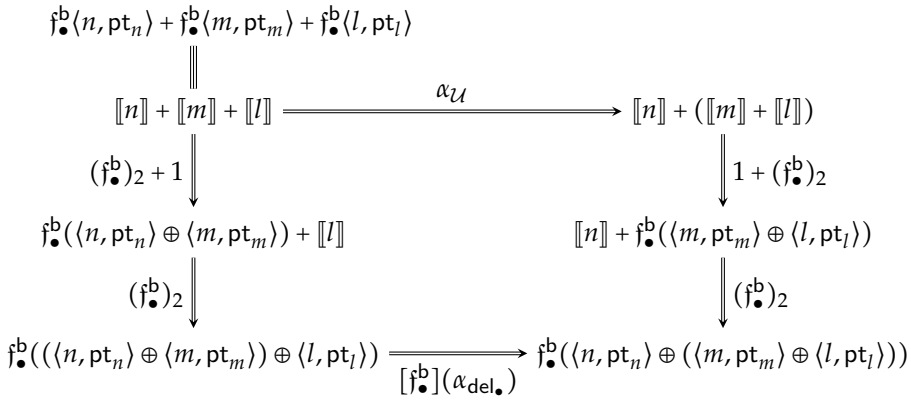
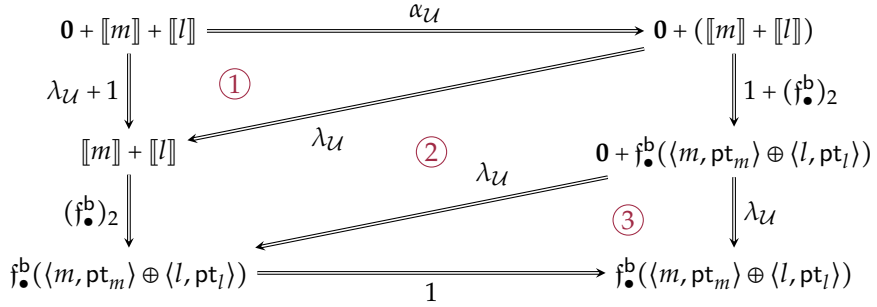


Figure 5.10: The requirement relative to the constructor tw in the inductive definition of the family $(\mathfrak{f}_\bullet)_2$ has, at its core, the displayed 2-path. The diagram commutes because the underlying functions of the two different compositions of equivalences corresponding to the paths in $\mathcal{U}$ bordering the 2-path are homotopic: they both send $\mathsf{inl}(\mathsf{inl}(\mathsf{inl}(x)))$ to $\mathsf{inl}(\mathsf{inl}(\mathsf{inl}(x)))$ for every $x : \mathfrak{f}^{\mathsf{b}}_n(a)$; $\mathsf{inl}(\mathsf{inl}(\mathsf{inr}(*)))$ to $\mathsf{inr}(*)$; $\mathsf{inl}(\mathsf{inr}(*))$ to $\mathsf{inl}(\mathsf{inr}(*))$; and $\mathsf{inr}(y)$ to $\mathsf{inl}(\mathsf{inl}(\mathsf{inr}(y)))$ for every $y : \mathfrak{f}^{\mathsf{b}}_m(b)$.

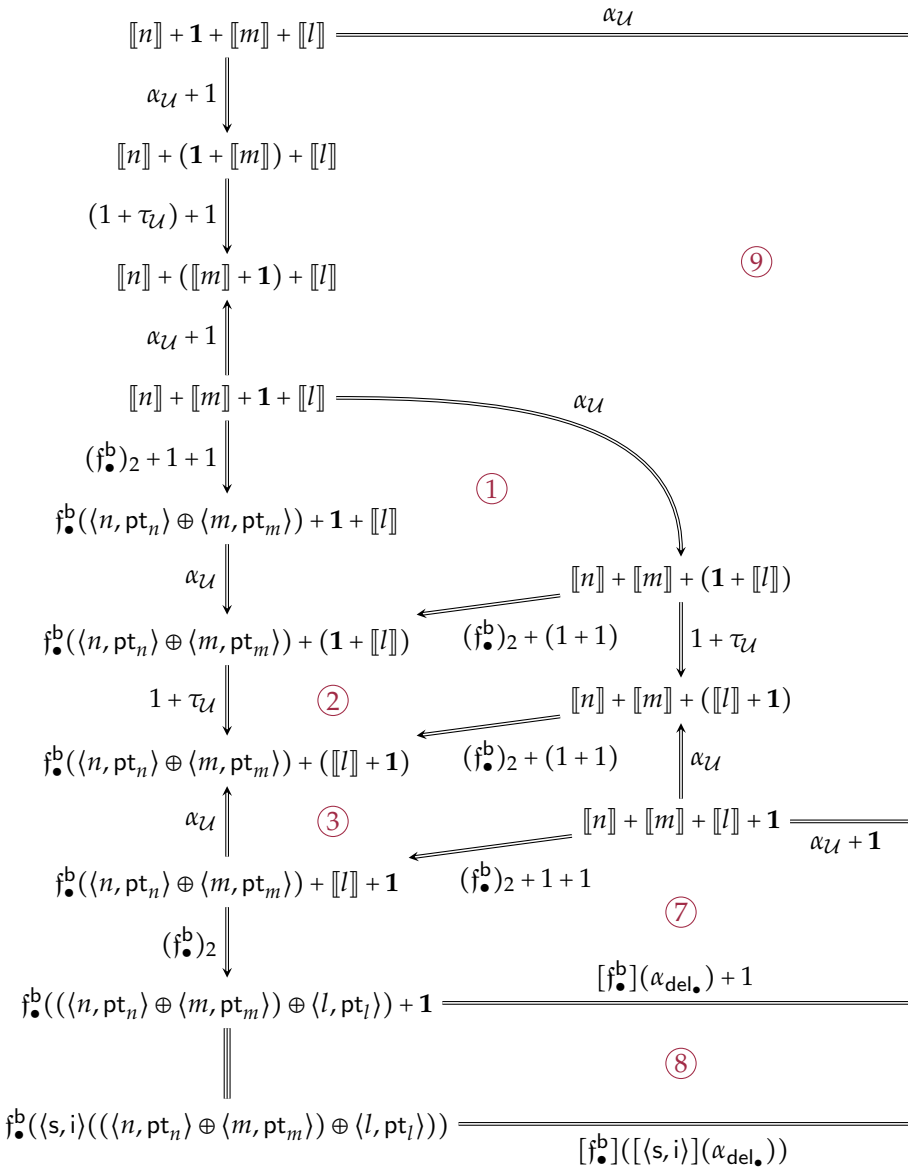


(a) The 2-path $(\mathfrak{f}^{\mathsf{b}}_\bullet)_\alpha(\langle n, \mathsf{pt}_n\rangle, \langle m, \mathsf{pt}_m\rangle, \langle l, \mathsf{pt}_l\rangle)$.



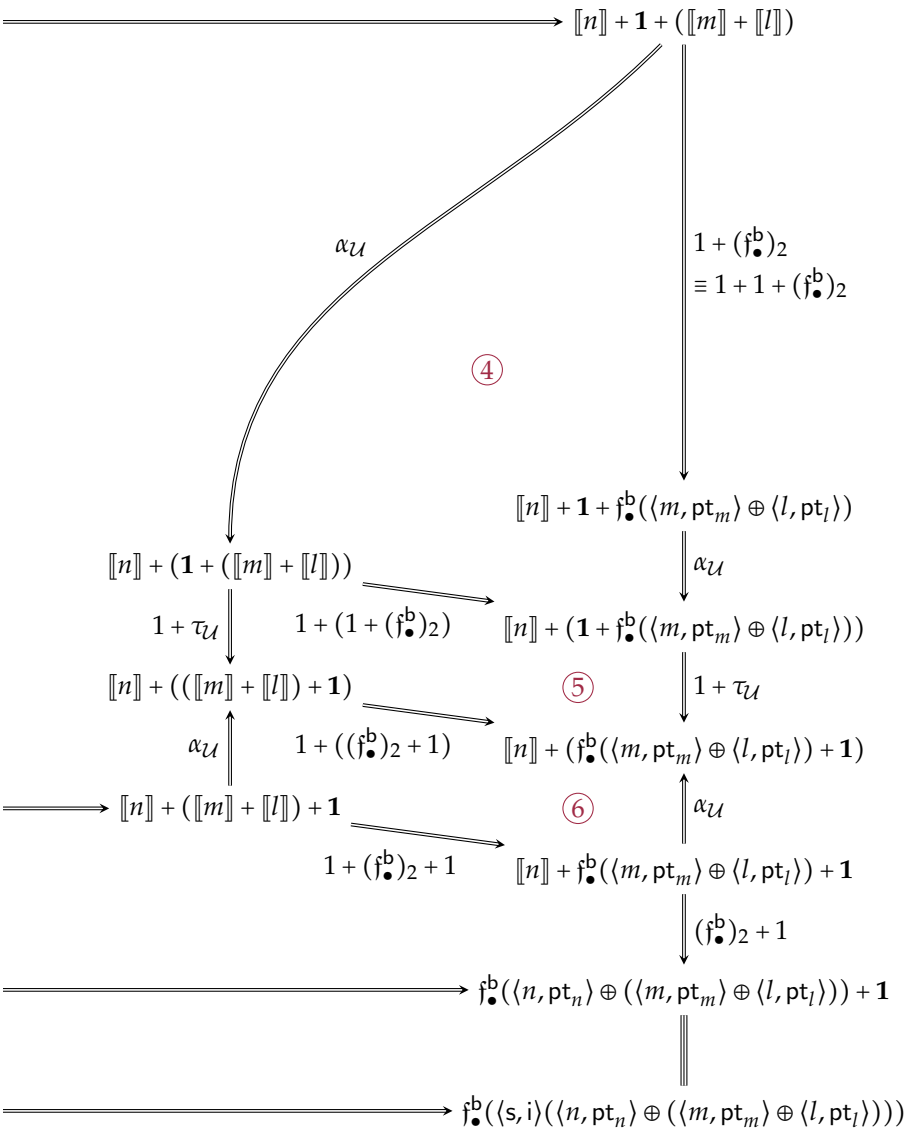
(b) The 2-path $(\mathfrak{f}^{\mathsf{b}}_\bullet)_\alpha(\langle 0, \mathsf{pt}_0\rangle, \langle m, \mathsf{pt}_m\rangle, \langle l, \mathsf{pt}_l\rangle)$. The 2-path (1) is an instance of the derived coherence diagram in Fig. 3.4a, but it can also be produced directly by examining the equivalences corresponding to the paths in $\mathcal{U}$; (2) is an instance of naturality of $\lambda_{\mathcal{U}}$; (3) is trivial.

Figure 5.11: Construction of $(\mathfrak{f}_\bullet)_\alpha$: **(a)** general case; **(b)** inductive case for pt_0; **(c)** inductive case for i.



(c) Construction of $(\mathfrak{f}_\bullet)_\alpha(\langle n+1, \mathsf{pt}_{n+1}\rangle, \langle m, \mathsf{pt}_m\rangle, \langle l, \mathsf{pt}_l\rangle)$. The 2-paths (1), (3), (4) and (6) are instances of naturality of $\alpha_{\mathcal{U}}$; (2) and (5) are instances of naturality of $\tau_{\mathcal{U}}$; (7) is derived from $(\mathfrak{f}_\bullet)_\alpha(\langle n, \mathsf{pt}_n\rangle, \langle m, \mathsf{pt}_m\rangle, \langle l, \mathsf{pt}_l\rangle)$; (8) can be obtained from the computation rule (5.41) of $\mathfrak{f}^{\mathsf{b}}$; (9) could be composed of coherence diagrams, but can also be found directly, as the paths bordering the diagram correspond to equivalences with homotopic underlying functions.

Figure 5.11: Continued.


$[\![n]\!] + \mathbf{1} + ([\![m]\!] + [\![l]\!])$
$\alpha_{\mathcal{U}}$
$1 + (\mathfrak{f}^{\mathsf{b}}_{\bullet})_2$
$\equiv 1 + 1 + (\mathfrak{f}^{\mathsf{b}}_{\bullet})_2$
④
$[\![n]\!] + \mathbf{1} + \mathfrak{f}^{\mathsf{b}}_{\bullet}(\langle m, \mathsf{pt}_m\rangle \oplus \langle l, \mathsf{pt}_l\rangle)$
$[\![n]\!] + ([\![m]\!] + [\![l]\!]) + \mathbf{1}$
$\alpha_{\mathcal{U}}$
$[\![n]\!] + (\mathbf{1} + \mathfrak{f}^{\mathsf{b}}_{\bullet}(\langle m, \mathsf{pt}_m\rangle \oplus \langle l, \mathsf{pt}_l\rangle))$
$1 + \tau_{\mathcal{U}}$
$1 + (1 + (\mathfrak{f}^{\mathsf{b}}_{\bullet})_2)$
$[\![n]\!] + (([\![m]\!] + [\![l]\!]) + \mathbf{1})$
⑤
$1 + \tau_{\mathcal{U}}$
$\alpha_{\mathcal{U}}$
$1 + ((\mathfrak{f}^{\mathsf{b}}_{\bullet})_2 + 1)$
$[\![n]\!] + (\mathfrak{f}^{\mathsf{b}}_{\bullet}(\langle m, \mathsf{pt}_m\rangle \oplus \langle l, \mathsf{pt}_l\rangle) + \mathbf{1})$
$[\![n]\!] + ([\![m]\!] + [\![l]\!]) + \mathbf{1}$
⑥
$\alpha_{\mathcal{U}}$
$1 + (\mathfrak{f}^{\mathsf{b}}_{\bullet})_2 + 1$
$[\![n]\!] + \mathfrak{f}^{\mathsf{b}}_{\bullet}(\langle m, \mathsf{pt}_m\rangle \oplus \langle l, \mathsf{pt}_l\rangle) + \mathbf{1}$
$(\mathfrak{f}^{\mathsf{b}}_{\bullet})_2 + 1$
$\mathfrak{f}^{\mathsf{b}}_{\bullet}(\langle n, \mathsf{pt}_n\rangle \oplus (\langle m, \mathsf{pt}_m\rangle \oplus \langle l, \mathsf{pt}_l\rangle)) + \mathbf{1}$
$\mathfrak{f}^{\mathsf{b}}_{\bullet}(\langle \mathsf{s}, \mathsf{i}\rangle(\langle n, \mathsf{pt}_n\rangle \oplus (\langle m, \mathsf{pt}_m\rangle \oplus \langle l, \mathsf{pt}_l\rangle)))$

Chapter 6

Directions for Further Research

This chapter presents additional work related to stating and proving coherence theorems for monoidal categories in HoTT. Although the content of this chapter is partial and not formalized, we believe that it can still have a place in this exposition, as it unveils possible problems in pursuing an alternative approach to the proof of coherence and, at the same time, it provides some ideas about how to further expand the results of coherence shown in the previous chapters.

6.1 Alternative Formulations of Coherence Statements[1]

While investigating possible ways to formalize coherence for monoidal and symmetric monoidal structures, we also considered an approach different from the one adopted in the previous chapters. The constructions FMG and FSMG given in this thesis are based on a 0-type X used as parameter for the HITs; in this sense, the types $\mathsf{FMG}(X)$ and $\mathsf{FSMG}(X)$ can be seen as spaces with a multiplication that is associative, unital and (for $\mathsf{FSMG}(X)$) symmetric up to 2-path coherence. As we saw in Theorem 5.104, coherence with respect to symmetric monoidal expressions behaves differently if such expressions contain repetitions of a term in the same product (as symmetric monoidal expressions without repetitions form a 0-type). For this reason, it is worth to examine the symmetric monoidal structure itself, leaving aside the type parameter X altogether.

In this section, we will briefly explain a strategy leading to a different formulation of a coherence statement for monoidal and symmetric monoidal structures. The underlying idea is to consider these structures using combinatorial species (in the sense of [Joy81]), building a layered notion of:

- species, here simply interpreted as functions from the subuniverse of 0-types to itself, possessing

[1] Joint work with Håkon R. Gylterud.

- a "monad" structure (with a 0-coherent unit and multiplication), together with
- a family of "equalities", versatile enough to describe (for a specific species) weak and strict monoidal and symmetric monoidal equality structures – which we aim to compare.

We will proceed to describe this notion more precisely, with the important *caveat* that the work presented here has not been completed. In this regard, since the purpose is to convey an idea rather than show a complete formalization, many details will be left out. In particular, we will work "informally" within the subuniverse Set of 0-types, without going into the trouble of expressing it as a Σ-type.

Definition 6.1. A **species** is a function $S : \mathsf{Set} \to \mathsf{Set}$. A **speciad** (species with a monad-like structure) consists of a species S together with:

- a unit term $\eta : S(\mathbf{1})$
- for every $A : \mathsf{Set}$, $s : S(A)$, $B : A \to \mathsf{Set}$ and $t : \prod(a : A) . S(B(a))$, a substitution (or multiplication) term $\mu_{s,t} : S(\Sigma(a : A) . B(a))$;
- families of pathovers witnessing associativity and left and right unitality (with respect to η) of the substitution. For example, for left unitality, given $A : \mathsf{Set}$ and $s : S(A)$, we demand that the substitution term

$$\mu_{\eta,\,(x \mapsto s)} : S(\Sigma(x : \mathbf{1}) . A) \tag{6.2}$$

"coincide" with s. The term s and the one in (6.2) are, evidently, of different types. However, there is an equivalence

$$e_\lambda : (\Sigma(x : \mathbf{1}) . A) \simeq A. \tag{6.3}$$

This induces, via univalence, a path $\mathsf{ua}(e_\lambda)$ in Set along which to transport, in the family S, the term in (6.2) to the type $S(A)$, where we demand a path to $s : S(A)$. Similarly, for right unitality, we require a pathover

$$(\mathsf{ua}(e_\rho))^S_* \left(\mu_{s,\,(a \mapsto \eta)}\right) = s \tag{6.4}$$

for every $s : S(A)$, where we used the equivalence $e_\rho : (\Sigma(a : A) . \mathbf{1}) \simeq A$. For associativity, given

$$\begin{aligned} &A : \mathsf{Set}, && s : S(A), \\ &B : A \to \mathsf{Set}, && t : \textstyle\prod(a : A) . S(B(a)), \\ &C : \textstyle\prod(a : A) . B(a) \to \mathsf{Set}, && u : \textstyle\prod(a : A) . \prod(b : B(a)) . S(C(a, b)), \end{aligned}$$

we consider the substitution terms

$$\mu_{s,\,(a\mapsto\mu_{t(a),u(a)})} : S(\Sigma\,(a:A)\,.\,\Sigma\,(b:B(a))\,.\,C(a,b)), \tag{6.5}$$

obtained by first acting on the family C and then on B, and

$$\mu_{(\mu_{s,t}),\,u^{\mathsf{uc}}} : S(\Sigma\,(x:\Sigma\,(a:A)\,.\,B(a))\,.\,C^{\mathsf{uc}}(x)), \tag{6.6}$$

obtained by first acting on B and then on C, where $C^{\mathsf{uc}} : (\Sigma\,(a:A)\,.\,B(a)) \to \mathsf{Set}$ is the uncurried version of C (i.e., $C^{\mathsf{uc}}\langle a,b\rangle :\equiv C(a,b)$, and similarly for u^{uc}). We then require a pathover

$$(\mathsf{ua}(e_\alpha))^S_*\left(\mu_{s,\,(a\mapsto\mu_{t(a),u(a)})}\right) = \mu_{(\mu_{s,t}),\,u^{\mathsf{uc}}}, \tag{6.7}$$

where the path in Set is obtained, by univalence, from the equivalence

$$e_\alpha : \Sigma\,(a:A)\,.\,\Sigma\,(b:B(a))\,.\,C(a,b) \;\simeq\; \Sigma\,(x:\Sigma\,(a:A)\,.\,B(a))\,.\,C^{\mathsf{uc}}(x). \tag{6.8}$$

Appropriate notions of *maps of species*, *maps of speciads* and *algebras over a speciad* can be defined, but we will not use them in this exposition.

Definition 6.9. The *magma* speciad is specified as follows:

- its underlying species $\mathsf{M} : \mathsf{Set} \to \mathsf{Set}$ is a family whose members are inductively defined as follows:

  $$\mathsf{M} ::= \mathsf{u} : \mathsf{M}(\mathbf{0}) \;\mid\; \mathsf{v} : \mathsf{M}(\mathbf{1}) \;\mid\; \mathsf{n} : \Pi\,(A_1, A_2 : \mathsf{Set})\,.\,\mathsf{M}(A_1) \to \mathsf{M}(A_2) \to \mathsf{M}(A_1 + A_2);$$

  we denote $\mathsf{n}_{A_1,A_2}(s_1,s_2)$ by $s_1 \boxplus s_2$;
- its unit is defined to be $\eta :\equiv \mathsf{v} : \mathsf{M}(\mathbf{1})$;
- its substitution $\mu_{s,t} : \mathsf{M}(\Sigma\,(a:A)\,.\,B(a))$, for $s : \mathsf{M}(A)$ and $t : \Pi\,(a:A)\,.\,\mathsf{M}(B(a))$, is defined by induction on s, as follows: the terms

  $$\mu_{\mathsf{u},t} : \mathsf{M}(\Sigma\,(x:\mathbf{0})\,.\,B(x))$$
  $$\mu_{\mathsf{v},t} : \mathsf{M}(\Sigma\,(x:\mathbf{1})\,.\,B(x))$$
  $$\mu_{s_1\boxplus s_2,t} : \mathsf{M}(\Sigma\,(a:A_1+A_2)\,.\,B(a))$$

  are obtained by transporting, respectively, the terms

  $$\mathsf{u} : \mathsf{M}(\mathbf{0})$$
  $$t(*) : \mathsf{M}(B(*))$$
  $$\mu_{s_1,t\circ\mathsf{inl}} \boxplus \mu_{s_2,t\circ\mathsf{inr}} : \mathsf{M}(\Sigma\,(a:A_1)\,.\,B(\mathsf{inl}(a)) + \Sigma\,(a:A_2)\,.\,B(\mathsf{inr}(a)))$$

  in the family M, along paths obtained via univalence from suitable equivalences ($\mathbf{0} \simeq \Sigma\,(x:\mathbf{0})\,.\,B(x)$, and so on);

- pathovers for associativity and unitality are obtainable, but omitted here.

The usefulness of the speciad M in our study lies essentially in the fact that, for a type $A : \mathsf{Set}$, a term $s : \mathsf{M}(A)$ yields easily a proof that the type A is finite – implicitly, because it shows that A is the coproduct of instances of the empty type and the unit type, in some order and with some choice of priority.[2] We want to reason about such coproducts as the combinatorial backbone of tree-like monoidal expressions, where the unit type acts as a placeholder for *data* (the kind of data injected by ι in FMG and FSMG), the empty type represents the unit in the monoidal structure, and the coproduct of types takes the place of the monoidal product. In this sense, for example, the term

$$((\mathsf{u} \boxplus \mathsf{v}) \boxplus (\mathsf{v} \boxplus \mathsf{u})) \boxplus \mathsf{v} : \mathsf{M}((\mathbf{0} + \mathbf{1}) + (\mathbf{1} + \mathbf{0}) + \mathbf{1})$$

embodies, from the point of view of combinatorics, all monoidal expressions

$$((e \otimes \iota(-)) \otimes (\iota(-) \otimes e)) \otimes \iota(-) : \mathsf{FMG}(-) \quad (\text{or } \mathsf{FSMG}(-)),$$

abstracting from the source of the data (the parameter of FMG).

The inductive definition of the magma species allows us to perform induction on monoidal expressions in a different way than FMG (and FSMG) could, since they are now untied from the paths and 2-paths that specify the rest of the monoidal structure; these will examined independently and encoded in families of types embodying suitable equivalence relations, effectively building *setoids*. These, as discussed in Section 3.2, are challenging to work with; here our aim is to avoid looking at quotients and simply compare equivalence relations, for which we need a suitable, general notion.

Definition 6.10. An **E-family** for a speciad $\langle S, \eta, \mu, \ldots \rangle$ is a family

$$E : \Pi\,(A_1, A_2 : \mathsf{Set})\,.\, S(A_1) \to S(A_2) \to (A_1 \simeq A_2) \to \mathsf{Set},$$

whose members will be denoted by $E_{s_1, s_2, \gamma}$, which is closed under identity, composition, inverses and substitution, and such that: the composition is associative; the identity behaves as a left and right unit for the composition; the inverse is an inverse operation with respect to the composition and the identity; and the substitution commutes with the group operations. That is, the following terms define the E-family:

[2] The converse, i.e. producing a term $s : \mathsf{M}(A)$ from a proof that A is finite, is not possible, as there is no canonical way of expressing a finite type as such a coproduct (this would be equivalent to finding a function $\|A \simeq [n]\| \to (A \simeq [n])$, for $n : \mathbb{N}$).

- $\iota_{A,\,s} : E_{s,\,s,\,\mathsf{id}_A}$ for every $A : \mathsf{Set}$, witnessing identity;
- $\circledcirc : E_{s_2,\,s_3,\,\gamma_2} \to E_{s_1,\,s_2,\,\gamma_1} \to E_{s_1,\,s_3,\,\gamma_2 \circ \gamma_1}$ (infix notation) for equivalences of sets $\gamma_1 : A_1 \simeq A_2$ and $\gamma_2 : A_2 \simeq A_3$, witnessing composition;
- $(-)^{-1} : E_{s_1,\,s_2,\,\gamma} \to E_{s_2,\,s_1,\,\gamma^{-1}}$, witnessing inverses;
- given arguments making the expressions $\mu_{s_1,\,t_1} : S(\Sigma\,(a : A_1)\,.\,B_1(a))$ and $\mu_{s_2,\,t_2} : S(\Sigma\,(a : A_2)\,.\,B_2(a))$ well-typed, and given an equivalence

  $$\langle \gamma, \delta \rangle : \Sigma\,(a : A_1)\,.\,B_1(a) \simeq \Sigma\,(a : A_2)\,.\,B_2(a)$$

  (in the notation of Definition 2.18), and terms

  $$e : E_{s_1,\,s_2,\,\gamma} \qquad \text{and} \qquad f : \Pi\,(a : A_1)\,.\,E_{t_1(a),\,t_2(\gamma(a)),\,\delta(a)},$$

  closure under substitution consists of a term

  $$\mu^E : E_{(\mu_{s_1,\,t_1}),\,(\mu_{s_2,\,t_2}),\,\langle \gamma, \delta \rangle},$$

  parsed as $\mu^E_{\gamma,\,e,\,\delta,\,f}$;
- families of paths witnessing the group laws for ι, $\circledcirc$ and $(-)^{-1}$, i.e.,

  $$\circledcirc^{\alpha} : (e_3 \circledcirc e_2) \circledcirc e_1 = e_3 \circledcirc (e_2 \circledcirc e_1),$$
  $$\circledcirc^{\lambda} : \iota_{A,\,s} \circledcirc e = e, \qquad \circledcirc^{-1} : e^{-1} \circledcirc e = \iota_{A,\,s},$$
  $$\circledcirc^{\rho} : e \circledcirc \iota_{A,\,s} = e, \qquad \circledcirc^{-\mathrm{r}} : e \circledcirc e^{-1} = \iota_{A,\,s},$$

  whenever such expressions are well-typed;
- families of paths witnessing commutativity of the substitution with the group operations, i.e.,

  $$\mu^{\circledcirc} : \mu^E_{(\gamma_2 \circ \gamma_1),\,(e_2 \circledcirc e_1),\,(a \mapsto \delta_2(a) \circ \delta_1(a)),\,(a \mapsto f_2(a) \circledcirc f_1(a))} = \mu^E_{\gamma_2,\,e_2,\,\delta_2,\,f_2} \circledcirc \mu^E_{\gamma_1,\,e_1,\,\delta_1,\,f_1},$$

  for arguments making the expression well-typed.

A speciad together with an E-family is called an **E-speciad**.

We will proceed to define two separate E-families for the magma speciad M, both referring to a monoidal structure for the monoidal expressions that the terms in the members of the family M are meant to represent.

Definition 6.11. Given $A : \mathsf{Set}$, a term $s : \mathsf{M}(A)$ provides an *order* $\leq_A : A \to A \to \mathsf{Prop}$ in A (where Prop is the subuniverse of (-1)-types; we will write $a_1 \leq_A a_2$ whenever the type $\leq_A(a_1, a_2)$ is inhabited), defined inductively on s:

- for $s \equiv \mathsf{u}$, the empty type $\mathbf{0}$ is ordered *ex falso*;
- for $s \equiv \mathsf{v}$, the unit type $\mathbf{1}$ is trivially ordered $(* \leq_{\mathbf{1}} *)$;
- for $s \equiv s_1 \boxplus s_2$, with $s_1 : \mathsf{M}(A_1)$, $s_2 : \mathsf{M}(A_2)$, the coproduct type $A_1 + A_2$ can be given an order by declaring:
  - $\mathsf{inl}(a_1) \leq_{A_1+A_2} \mathsf{inl}(a_1')$ if $a_1 \leq_{A_1} a_1'$, for any $a_1, a_1' : A_1$;
  - $\mathsf{inr}(a_2) \leq_{A_1+A_2} \mathsf{inr}(a_2')$ if $a_2 \leq_{A_2} a_2'$, for any $a_2, a_2' : A_2$;
  - $\mathsf{inl}(a_1) \leq_{A_1+A_2} \mathsf{inr}(a_2)$ for every $a_1 : A_1$ and $a_2 : A_2$.

It is easy to see that $\leq_A$ is a reflexive and transitive relation.

We define an E-family E^{wk} for the magma speciad by declaring, for every equivalence $\gamma : A_1 \simeq A_2$,

$$E^{\mathsf{wk}}_{s_1,\, s_2,\, \gamma} :\equiv \Pi\,(x, y : A_1)\,.\,(x \leq_{A_1} y) \to (\gamma(x) \leq_{A_2} \gamma(y));$$

that is, $E^{\mathsf{wk}}_{s_1,\, s_2,\, \gamma}$ is the type encoding that γ is *order-preserving* ("O.P."). This definition satisfies all the requirements of an E-family, once noticed: that the identity equivalence is O.P.; that the composition of O.P. equivalences is O.P.; that the inverse of an O.P. equivalence is O.P.; that an O.P. equivalence on the base type of a fibration together with a pointwise O.P. equivalence on its fibers induce an O.P. equivalence on the total space; that E^{wk} is a family of propositions. The resulting E-speciad will be called a *magma with a weak monoidal structure*.

The E-family E^{wk} defined above serves the purpose of placing in the same equivalence relation those equivalences between coproducts of instances of the empty type and the unit type which do not "scramble" the instances of the unit type. For example, relatively to the distinct equivalences $\mathsf{id}_{\mathbf{1}+\mathbf{1}}$ and $\tau : \mathbf{1}+\mathbf{1} \simeq \mathbf{1}+\mathbf{1}$, we see that the type $E^{\mathsf{wk}}_{\mathsf{v}\boxplus\mathsf{v},\, \mathsf{v}\boxplus\mathsf{v},\, \mathsf{id}_{\mathbf{1}+\mathbf{1}}}$ is inhabited, while $E^{\mathsf{wk}}_{\mathsf{v}\boxplus\mathsf{v},\, \mathsf{v}\boxplus\mathsf{v},\, \tau}$ is not (because τ is not O.P.).

Definition 6.12. We define an E-family E^{str} for the magma speciad as a higher inductive family of types with the following constructors:

- all terms specified in Definition 6.10 are constructors;
- terms

$$e^{\alpha} : E^{\mathsf{str}}_{(\mathsf{v}\boxplus\mathsf{v})\boxplus\mathsf{v},\, \mathsf{v}\boxplus(\mathsf{v}\boxplus\mathsf{v})),\, \alpha} \qquad e^{\lambda} : E^{\mathsf{str}}_{\mathsf{u}\boxplus\mathsf{v},\, \mathsf{v},\, \lambda} \qquad e^{\rho} : E^{\mathsf{str}}_{\mathsf{v}\boxplus\mathsf{u},\, \mathsf{v},\, \rho}$$

  for $\alpha : \mathbf{1}+\mathbf{1}+\mathbf{1} \simeq \mathbf{1}+(\mathbf{1}+\mathbf{1})$, $\lambda : \mathbf{0}+\mathbf{1} \simeq \mathbf{1}$ and $\rho : \mathbf{1}+\mathbf{0} \simeq \mathbf{1}$ the "canonical" equivalences, are constructors;

- path constructors related to the coherence diagrams for a monoidal structure. We leave the complete expressions unspecified here, as the substitution machinery needed to make them type-check is rather involved, and this definition has not been formalized on a proof assistant yet. The general idea is that two pairs of equivalences,

$$h_1^{\pentagon},\ h_2^{\pentagon} : \mathbf{1}+\mathbf{1}+\mathbf{1}+\mathbf{1} \simeq \mathbf{1}+(\mathbf{1}+(\mathbf{1}+\mathbf{1})) \quad \text{and}$$
$$h_1^{\triangledown},\ h_2^{\triangledown} : \mathbf{1}+\mathbf{0}+\mathbf{1} \simeq \mathbf{1}+\mathbf{1},$$

  can be obtained from α, λ, ρ (and substitution tricks); these mimic, in their definitions, the different ways to travel around the coherence diagrams $\pentagon$ and $\triangledown$ in a monoidal structure, and they are provably equal (i.e., there are paths $p^{\pentagon} : h_1^{\pentagon} = h_2^{\pentagon}$ and $p^{\triangledown} : h_1^{\triangledown} = h_2^{\triangledown}$). Corresponding terms

$$e_1^{\pentagon} : E^{\mathsf{str}}_{((\mathsf{v}\boxplus\mathsf{v})\boxplus\mathsf{v})\boxplus\mathsf{v},\ \mathsf{v}\boxplus(\mathsf{v}\boxplus(\mathsf{v}\boxplus\mathsf{v})),\ h_1^{\pentagon}} \qquad e_2^{\pentagon} : E^{\mathsf{str}}_{((\mathsf{v}\boxplus\mathsf{v})\boxplus\mathsf{v})\boxplus\mathsf{v},\ \mathsf{v}\boxplus(\mathsf{v}\boxplus(\mathsf{v}\boxplus\mathsf{v})),\ h_2^{\pentagon}}$$
$$e_1^{\triangledown} : E^{\mathsf{str}}_{(\mathsf{v}\boxplus\mathsf{u})\boxplus\mathsf{v},\ \mathsf{v}\boxplus\mathsf{v},\ h_1^{\triangledown}} \qquad e_2^{\triangledown} : E^{\mathsf{str}}_{(\mathsf{v}\boxplus\mathsf{u})\boxplus\mathsf{v},\ \mathsf{v}\boxplus\mathsf{v},\ h_2^{\triangledown}}$$

  can be built from the term constructors. The path constructors defining E^{str} are then the pathovers

$$\left(p^{\pentagon}\right)_*^{E^{\mathsf{str}}_{((\mathsf{v}\boxplus\mathsf{v})\boxplus\mathsf{v})\boxplus\mathsf{v},\ \mathsf{v}\boxplus(\mathsf{v}\boxplus(\mathsf{v}\boxplus\mathsf{v}))}}\left(e_1^{\pentagon}\right) = e_2^{\pentagon} \quad \text{and}$$
$$\left(p^{\triangledown}\right)_*^{E^{\mathsf{str}}_{(\mathsf{v}\boxplus\mathsf{u})\boxplus\mathsf{v},\ \mathsf{v}\boxplus\mathsf{v}}}\left(e_1^{\triangledown}\right) = e_2^{\triangledown}.$$

In other words, E^{str} is the "E-family closure" of the minimal terms specifying associativity, unitality and coherence. The resulting E-speciad will be called a *magma with a strict monoidal structure*.

Now that we defined a "weak" and a "strict" E-family for the magma speciad, the natural goal is to compare them.

Statement 6.13 (Coherence for monoidal structures)**.** *The type families E^{wk} and E^{str} in Definitions 6.11 and 6.12 are equivalent, i.e., for every A_1, $A_2 : \mathsf{Set}$, $s_1 : \mathsf{M}(A_1)$, $s_2 : \mathsf{M}(A_2)$ and $\gamma : A_1 \simeq A_2$, there is an equivalence*

$$E^{wk}_{s_1,\,s_2,\,\gamma} \simeq E^{str}_{s_1,\,s_2,\,\gamma}.$$

This corresponds roughly to the statement: "Every order-preserving equivalence γ between coproducts of instances of the empty type and the unit type can be expressed as a composition of coproducts of instances of the equivalences α, λ

and ρ'', which indeed represents the combinatorial core of coherence for monoidal categories. One notices immediately that, in order to prove Statement 6.13, it is enough to show that E^{str} is a family of (-1)-types; this exposes a very clear parallel to the statement of coherence for monoidal groupoids discussed in Chapter 3, which entails showing that all identity types between terms in $\mathsf{FMG}(X)$ are (-1)-types (equivalently, that $\mathsf{FMG}(X)$ is a 0-type).

Monoidal structures as interpreted in this framework concern only O.P. equivalences, since associativity and unitality do not disturb the order of the data. For symmetric monoidal structures, we allow the positions for data to trade places; this motivates the following definition.

Definition 6.14. The E-family $E^{\mathsf{s\text{-}wk}}$ for the magma speciad is defined to be the constant family

$$E^{\mathsf{s\text{-}wk}}_{s_1,\, s_2,\, \gamma} :\equiv \mathbf{1}$$

for every A_1, $A_2 : \mathsf{Set}$, $s_1 : \mathsf{M}(A_1)$, $s_2 : \mathsf{M}(A_2)$ and $\gamma : A_1 \simeq A_2$. All conditions required in the definition of an E-family are trivially met. The ensuing E-speciad is dubbed *magma with a symmetric weak monoidal structure*.

Predictably, a corresponding strict structure can be defined, and a statement of coherence for symmetric monoidal structures follows.

Definition 6.15. The E-family $E^{\mathsf{s\text{-}str}}$ for the magma speciad is specified similarly to E^{str} in Definition 6.12, accounting also for symmetries (that is, with the addition of a constructor $e^{\tau} : E^{\mathsf{s\text{-}str}}_{\mathsf{v}\boxplus\mathsf{v},\, \mathsf{v}\boxplus\mathsf{v},\, \tau}$ with $\tau : \mathbf{1}+\mathbf{1} \simeq \mathbf{1}+\mathbf{1}$ the nontrivial equivalence, and path constructors relative to the coherence diagrams). The resulting E-speciad is called *magma with a symmetric strict monoidal structure*.

Statement 6.16 (Coherence for symmetric monoidal structures)**.** *The type families $E^{\mathsf{s\text{-}wk}}$ and $E^{\mathsf{s\text{-}str}}$ in Definitions 6.14 and 6.15 are equivalent, i.e., for every A_1, $A_2 : \mathsf{Set}$, $s_1 : \mathsf{M}(A_1)$, $s_2 : \mathsf{M}(A_2)$ and $\gamma : A_1 \simeq A_2$, there is an equivalence*

$$E^{\mathsf{s\text{-}wk}}_{s_1,\, s_2,\, \gamma} \simeq E^{\mathsf{s\text{-}str}}_{s_1,\, s_2,\, \gamma}.$$

In other words, the claim above states: "Every equivalence γ between coproducts of instances of the empty type and the unit type can be expressed as a composition of coproducts of instances of the equivalences α, λ, ρ and τ", i.e., these four "generate" all equivalences.

The coherence theorems in Statements 6.13 and 6.16 were not proved, and their formalization remains, at the present time, at a very preliminary stage. We speculate that the complexity of their proofs will match that of the proofs presented

in the previous chapters, with the added burden of having to work, already from the very definition of a speciad, with paths over paths obtained via univalence. While acknowledging that this approach to coherence needs further research, we are also aware that similar problems of establishing a type-theoretic language to address monoidal structures and coherence have been investigated by other research groups [see e.g. Fio+08; FS19; Sav20].

6.2 Other Monoidal Structures

Braided Monoidal Groupoids

Another flavour of monoidality often considered for a product in a category is braided monoidality. A *braided* monoidal structure on a category is a weaker variant of a symmetric monoidal structure, where the symmetry natural isomorphism is not the inverse of itself. In the language of this thesis, a braided monoidal structure on a type M consists of a unit term $e_M : M$, a function $\otimes_M : M \to M \to M$, families of paths α_M, λ_M, ρ_M and τ_M and of 2-paths $\pentagon_M$, $\triangledown_M$ and $\hexagon_M$ as in Definition 4.1, and an ulterior coherence diagram $\hexagon'_M$ shown in Fig. 4.4c (note that $\hexagon\!\!\!\!-_M$ is not present in the definition, nor it can be derived).

Results parallel to those presented for symmetric monoidal groupoids in Chapter 4 and, partially, Chapter 5 also hold for braided monoidal groupoids:

- for $X : \mathcal{U}$, we can define a HIT $\mathsf{FBMG}(X)$, with constructors given by the definition of a braided monoidal structure; the ensuing construction $\mathsf{FBMG} : \mathcal{U} \to \mathcal{U}$ is free;
- the type $\mathsf{FBMG}(X)$ is equivalent to a HIT $\mathsf{blist}(X)$ of "braided lists", via braided monoidal functors. the constructors of $\mathsf{blist}(X)$ are the same as those of $\mathsf{slist}(X)$, save for double; these stem from the following presentation of the braid groups B_n on n strands: the groups B_0 and B_1 are trivial, while

$$\mathsf{B}_{n+2} := \frac{(a_1, \dots, a_{n+1})}{a_{i+1}a_ia_{i+1} = a_ia_{i+1}a_i, \quad a_ia_j = a_ja_i \ \text{ for } |i-j| \geq 2}$$

  (cf. (4.10) for the presentation of symmetric groups, which has the additional relations $a_i^2 = 1$). This proof of normalisation – i.e., a (braided monoidal) equivalence $\mathsf{FBMG}(X) \simeq \mathsf{blist}(X)$ – follows closely the one given in Chapter 4 for symmetric monoidal expressions;

- similarly, an indexed family $\mathsf{bdel} : \mathbb{N} \to \mathcal{U}$ of HITs encoding the deloopings of the braid groups can be defined and its total space $\mathsf{bdel}_\bullet$ can be proved equivalent to $\mathsf{blist}(\mathbf{1})$.

Given that the constructions $\mathsf{FBMG}(X)$, $\mathsf{blist}(X)$ and $\mathsf{bdel}_\bullet$ and the relevant proofs mentioned above would go along the lines of those in Chapter 4 and Chapter 5 for symmetric monoidal groupoids, a more detailed dissertation would be redundant.

We did not find an easy description in HoTT of the classifying spaces $\mathcal{B}\mathrm{B}_n$ of the braid groups. Referring to the content of Section 6.1, we do not know of a definition of an E-family $E^{\mathsf{b\text{-}wk}}$ to describe a magma with a braided weak monoidal structure either, revealing that the complexity of braided structures is harder to formulate than that of symmetric structures.

Distributive Monoidal Groupoids

A *distributive* (also *bimonoidal* or *rig*) structure on a category consists of two symmetric monoidal structures interacting with each other, akin to the two operations in a semiring. In the language of this thesis, a bimonoidal structure on a type M is given by:

- a unit term $e_M : M$ and a null term $v_M : M$;
- functions $\otimes_M$ and $\oplus_M : M \to M \to M$ (respectively, monoidal product and monoidal coproduct);
- families of paths witnessing associativity, unitality (with respect to e_M and v_M respectively) and symmetry of the monoidal product and coproduct;
- families of paths

$$\kappa^{\mathsf{l}}_M : \Pi\,(b : M)\,.\,v_M \otimes_M b = v_M,$$
$$\kappa^{\mathsf{r}}_M : \Pi\,(a : M)\,.\,a \otimes_M v_M = v_M,$$

  witnessing absorption, and families of paths

$$\delta^{\mathsf{l}}_M : \Pi\,(a, b, c : M)\,.\,(a \oplus_M b) \otimes_M c = (a \otimes_M c) \oplus (b \otimes_M c),$$
$$\delta^{\mathsf{r}}_M : \Pi\,(a, b, c : M)\,.\,a \otimes_M (b \oplus_M c) = (a \otimes_M b) \oplus (a \otimes_M c),$$

  witnessing distributivity of the monoidal product over the monoidal coproduct;
- a large number of families of 2-paths (coherence diagrams), which are described in detail in [Lap72].

Once defined a HIT of free distributive monoidal expressions, a possible HIT of normal forms could be based on $\mathsf{slist}(\mathsf{slist}(X))$, with additional higher constructors. A term in such a type is an unordered list of unordered lists, such as

$$(x :: y :: z :: \mathsf{nil}) :: (x :: y :: \mathsf{nil}) :: \mathsf{nil},$$

which ought to be the normal form, for instance, of

$$(\iota(x) \otimes (\iota(y) \otimes \iota(z))) \oplus (\iota(x) \otimes \iota(y)),$$

i.e., a list of lists represents a coproduct of products. The null term in $\mathsf{slist}(\mathsf{slist}(X))$ is the empty list nil, while the unit term is the list $\mathsf{nil} :: \mathsf{nil}$ containing the empty list. Then the already defined operation $+\!\!+$ (appending symmetric lists, at the exterior level) is the monoidal *co*product, while a monoidal product $\times : \mathsf{slist}(\mathsf{slist}(X)) \to \mathsf{slist}(\mathsf{slist}(X)) \to \mathsf{slist}(\mathsf{slist}(X))$ needs to be defined as a new operation. Leaving aside all details on higher constructors, in order to match with the distributivity of $\otimes$ over $\oplus$, such a function should compute, for every $L_1, L_2 : \mathsf{slist}(\mathsf{slist}(X))$ and $l : \mathsf{slist}(X)$:

$$L_1 \times \mathsf{nil} \equiv \mathsf{nil}, \qquad L_1 \times (l :: L_2) \equiv (L_1 \ltimes l) +\!\!+ (L_1 \times L_2),$$

where the auxiliary function $\ltimes : \mathsf{slist}(\mathsf{slist}(X)) \to \mathsf{slist}(X) \to \mathsf{slist}(\mathsf{slist}(X))$, also defined by induction on the first argument, should compute, on 0-constructors:

$$\mathsf{nil} \ltimes l \equiv \mathsf{nil}, \qquad (l' :: L) \ltimes l \equiv (l' +\!\!+ l) :: (L \ltimes l).$$

Contrary to braided monoidal structures, normalisation of distributive monoidal expressions seems to be fundamentally different from normalisation of symmetric monoidal expressions, as the distributivity δ^{l} and δ^{r} involve duplication of terms, and the diagonal function

$$\Delta :\equiv (a \mapsto \langle a, a \rangle) : M \to (M \times M)$$

is not invertible. Further studies are then needed to formulate and formalize normalisation of such expressions in HoTT. As for the type of "unordered, distributive vectors" on a (0-)type X, the obvious guess is to "iterate" the construction based on $\mathcal{BS}_\bullet$:

$$\begin{aligned}\Sigma\,(A : \mathcal{U})\,.\,\Big((\Sigma\,(n : \mathbb{N})\,.\,\|A \simeq [n]\|) \times \\ \Big(\Sigma\,(B : A \to \mathcal{U})\,.\,(\Pi\,(a : A)\,.\,\Sigma\,(m : \mathbb{N})\,.\,\|B(a) \simeq [m]\|) \times \\ \Sigma\,(a : A)\,.\,B(a) \to X\Big)\Big),\end{aligned}$$

i.e., these are functions $\Sigma\,(a : A)\,.\,B(a) \to X$ for a finite type A and a family of finite types $B : A \to \mathcal{U}$.

Appendix A

Formalization in Coq

The computer verification has been written using the HoTT library for the proof assistant Coq [Hoq].[1] We include here only a few selected fragments; the full formalization is accessible online.[2]

The structure of the files in the formalization is displayed in Fig. A.1; a description of their content is shown in Table A.2.

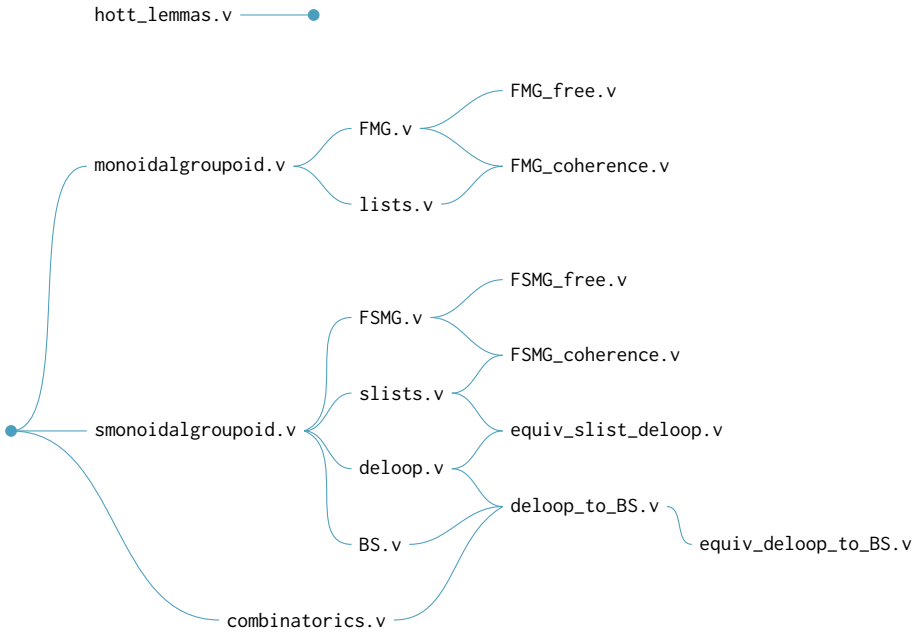


Figure A.1: Structure of the files in the Coq formalization (the dependency relation is to be read left-to-right).

[1] The formalization runs on commit `68774877142adbd435ea5013c5f201f3ec6ff66a` (February 14th, 2020) of the HoTT library.

[2] https://github.com/spiceghello/FSMG

File	Content	Chapter
`hott_lemmas.v`	Path algebra; function extensionality algebra.	2
`monoidalgroupoid.v`	Monoidal structures, groupoids and functors; derived coherence diagrams in a monoidal groupoid.	3 (A.1)
`lists.v`	Lists over 0-types are 0-types; a monoidal structure on $\mathsf{list}(X)$.	
`FMG.v`	Definition of the HIT construction FMG.	
`FMG_free.v`	Freeness of FMG.	
`FMG_coherence.v`	A monoidal equivalence $\mathsf{FMG}(X) \simeq \mathsf{list}(X)$; notes on the normalizing functor in [BD96; Bey97].	
`smonoidalgroupoid.v`	Symmetric monoidal structures, groupoids and functors; derived coherence diagrams in a symmetric monoidal groupoid.	4 (A.2)
`slists.v`	Definition of the HIT construction slist and its symmetric monoidal structure.	
`FSMG.v`	Definition of the HIT construction FSMG.	
`FSMG_free.v`	Freeness of FSMG.	
`FSMG_coherence.v`	A symmetric monoidal equivalence $\mathsf{FSMG}(X) \simeq \mathsf{slist}(X)$.	
`BS.v`	Finite types $\mathcal{BS}_\bullet$ and their symmetric monoidal structure.	5 (A.3)
`deloop.v`	Definition of the HIT construction $\mathsf{del}_{(-)}$ and a symmetric monoidal structure on $\mathsf{del}_\bullet$.[†]	
`equiv_slist_deloop.v`	A symmetric monoidal equivalence $\mathsf{slist}(\mathbf{1}) \simeq \mathsf{del}_\bullet$.	
`combinatorics.v`	Definitions of add, incr, $e_{\vert A}$, ω and relevant lemmata.	
`deloop_to_BS.v`	Definition of the family of functions $\mathfrak{f} : \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \to \mathcal{BS}_n$; sketch of the proof that $\mathfrak{f}_\bullet$ is a symmetric monoidal functor.	
`equiv_deloop_to_BS.v`	Proof that $\mathfrak{f}_n$ is an equivalence for every $n : \mathbb{N}$, under the unformalized assumption described in the thesis.	

Table A.2: Description of the files in the Coq formalization, referring to the chapter in which the content appears in this thesis and to the section of this appendix where it is presented.
[†] The symmetric monoidal structure on $\mathsf{del}_\bullet$ is short of the coherence diagrams $\hexagon$ and $\diamond$, which can however be deduced since all its coherence diagrams are proved similarly to the ones of $\mathsf{slist}(X)$.

A.1 Coherence for Monoidal Groupoids

For the definitions of monoidal groupoids, monoidal functors, monoidal natural isomorphisms and free functors, we use classes instead of Σ-types for easy access to their components and for type coercions. Below is the implementation of Definition 3.11.

```
Section MonoidalStructure.

Context {X : Type} (e : X) (m : X -> X -> X).

Definition IsAssociative : Type
  := forall a b c : X, m (m a b) c = m a (m b c).

Definition IsLeftUnital : Type
  := forall b : X, m e b = b.

Definition IsRightUnital : Type
  := forall a : X, m a e = a.

Definition IsPentagonCoherent (alpha : IsAssociative) : Type
  := forall a b c d : X,
      alpha (m a b) c d @ alpha a b (m c d)
      = ap011 m (alpha a b c) (idpath d) @ alpha a (m b c) d
        @ ap011 m (idpath a) (alpha b c d).

Definition IsTriangleCoherent (alpha : IsAssociative) (lambda : IsLeftUnital)
  (rho : IsRightUnital) : Type
  := forall a b : X,
      alpha a e b @ ap011 m (idpath a) (lambda b)
      = ap011 m (rho a) (idpath b).

End MonoidalStructure.

Class MonoidalGroupoid := {
  mgcarrier : Type;
  mgtrunc : IsTrunc 1 mgcarrier;
  mg_e : mgcarrier;
  mg_m : mgcarrier -> mgcarrier -> mgcarrier;
  mg_alpha : IsAssociative mg_m;
  mg_lambda : IsLeftUnital mg_e mg_m;
  mg_rho : IsRightUnital mg_e mg_m;
  mg_pentagon : IsPentagonCoherent mg_m mg_alpha;
  mg_triangle : IsTriangleCoherent mg_e mg_m mg_alpha mg_lambda mg_rho
  }.
```

We can use the same name for a monoidal groupoid and its underlying type:

```
Coercion mgcarrier : MonoidalGroupoid >-> Sortclass.
```

We give a short name to the function $[- \otimes_M -]$ (notation as in Remark 2.65), using `ap011` from the library:

```
Definition mg_mm {M : MonoidalGroupoid} {a b a' b'} (pa : a = a') (pb : b = b')
  : mg_m a b = mg_m a' b'
  := ap011 mg_m pa pb.
```

Naturality of α_M, λ_M and ρ_M (Lemma 3.15) follows by path induction; for example:

```
Definition alpha_natural {a b c a' b' c'}
  (pa : a = a') (pb : b = b') (pc : c = c')
  : mg_alpha a b c @ mg_mm pa (mg_mm pb pc)
    = mg_mm (mg_mm pa pb) pc @ mg_alpha a' b' c'.
Proof.
  induction pa, pb, pc.
  exact (concat_p1 _ @ (concat_1p _)^).
Defined.
```

proves naturality of α_M. The first line of the proof performs induction on the three paths; since refl ⊗ refl ≡ refl, it is enough to show that

$$\alpha_M(a,b,c) \cdot \mathsf{refl}_{a\otimes(b\otimes c)} = \mathsf{refl}_{(a\otimes b)\otimes c} \cdot \alpha_M(a,b,c),$$

which is done on the second line of the proof, using library-defined terms for unitality of concatenation of paths (Definition 2.48).

Much of the path algebra required to construct 2-paths in the thesis is invisible in the figures, in which we treat, for instance, concatenation of paths as if it were judgmentally associative and unital (see Remark 3.8, Remark 2.51). In the formalization, of course, a different (and harder) kind of work is required. As an example, we present the construction of the derived coherence diagram in Fig. 3.4a. The numbers on the left refer to the 2-paths in Fig. 3.9a.

```
Definition alpha_lambda (a b : M)
  : mg_alpha mg_e a b @ mg_lambda (mg_m a b) = mg_mm (mg_lambda a) (idpath b).
Proof.
1 apply (cancelL (mg_lambda _) _ _); refine (concat_p_pp _ _ _ @ _);
    refine (whiskerR (lambda_natural (mg_alpha mg_e a b)) _
    @ concat_pp_p _ _ _ @ _).
2 refine (whiskerL _ (lambda_natural (mg_lambda (mg_m a b)))
    @ concat_p_pp _ _ _ @ _).
3 refine (_ @ (lambda_natural (mg_mm (mg_lambda a) (idpath b)))^);
    apply whiskerR.
4 apply (cancelL (mg_alpha _ _ _) _ _); refine (concat_p_pp _ _ _ @ _);
    refine (_ @ (alpha_natural (idpath mg_e) (mg_lambda a) (idpath b))^);
    refine (concat_pp_p _ _ _ @ _).
8 apply (cancelL (mg_mm (mg_alpha _ _ _) idpath) _ _);
    refine (concat_p_pp _ _ _ @ concat_p_pp _ _ _ @ _ @ concat_pp_p _ _ _);
    refine (whiskerR (mg_pentagon mg_e mg_e a b)^ _ @ concat_pp_p _ _ _ @ _).
7 refine (whiskerL _ (mg_triangle mg_e (mg_m a b)) @ _).
6 refine (alpha_natural (mg_rho mg_e) idpath idpath @ _); apply whiskerR.
```

```
5 refine (_ @ (ap011_pqpq mg_m (mg_alpha mg_e mg_e a)
    (mg_mm idpath (mg_lambda a)) idpath idpath)^);
    exact (mg_mmm (mg_triangle _ _)^ idpath).
Defined.
```

The term `mg_mmm` is $[[- \otimes -]]$, while `ap011_pqpq` is the interchange law in (2.64).

Monoidal functors and monoidal natural isomorphisms (described in Definitions 3.17 and 3.18) are implemented as follows.

```
Class MonoidalFunctor (A B : MonoidalGroupoid) := {
  mg_f : A -> B;
  mg_f0 : mg_e = mg_f mg_e;
  mg_f2 : forall a b : A, mg_m (mg_f a) (mg_f b) = mg_f (mg_m a b);
  mg_dalpha : forall a b c : A,
    mg_alpha (mg_f a) (mg_f b) (mg_f c) @ (mg_mm idpath (mg_f2 b c) @ mg_f2 a (mg_m b c))
    = (mg_mm (mg_f2 a b) idpath @ mg_f2 (mg_m a b) c) @ ap mg_f (mg_alpha a b c);
  mg_dlambda : forall b : A,
    mg_lambda (mg_f b) = mg_mm mg_f0 idpath @ mg_f2 mg_e b @ ap mg_f (mg_lambda b);
  mg_drho : forall a : A,
    mg_rho (mg_f a) = mg_mm idpath mg_f0 @ mg_f2 a mg_e @ ap mg_f (mg_rho a)
  }.
Coercion mg_f : MonoidalFunctor >-> Funclass.
```

```
Class MonoidalNatIso {A B} (F G : MonoidalFunctor A B) := {
  mg_nt : F == G;
  mg_nt0 : @mg_f0 _ _ F @ mg_nt mg_e = @mg_f0 _ _ G;
  mg_nt2 : forall a b : A,
    @mg_f2 _ _ F a b @ mg_nt (mg_m a b)
    = ap011 mg_m (mg_nt a) (mg_nt b) @ @mg_f2 _ _ G a b
  }.
```

Finally, functors and free functors are also given as classes. In the following excerpt we show the implementation of Definition 3.23 and Definition 3.24; the terms `MonoidalFunctor_id` and `MonoidalFunctor_comp` are, respectively, the identity monoidal functor and the composition of monoidal functors, whose construction we omit.

```
Class IsFunctor (F : forall X : Type, IsHSet X -> MonoidalGroupoid) := {
  F_arr : forall (X Y : Type) (T_X : IsHSet X) (T_Y : IsHSet Y) (f : X -> Y),
          MonoidalFunctor (F X T_X) (F Y T_Y);
  F_id : forall (X : Type) (T_X : IsHSet X),
          MonoidalNatIso (F_arr X X T_X T_X (fun x => x)) (MonoidalFunctor_id (F X T_X));
  F_comp : forall (X Y Z : Type) (T_X : IsHSet X) (T_Y : IsHSet Y) (T_Z : IsHSet Z)
            (g : Y -> Z) (f : X -> Y),
            MonoidalNatIso
              (MonoidalFunctor_comp (F_arr Y Z T_Y T_Z g) (F_arr X Y T_X T_Y f))
              (F_arr X Z T_X T_Z (g o f))
  }.

Class IsFreeFunctor (F : forall X : Type, IsHSet X -> MonoidalGroupoid) := {
  free_functor : IsFunctor F;
```

```
  Phi : forall (X : Type) (T_X : IsHSet X)
        (M : MonoidalGroupoid) (G : MonoidalFunctor (F X T_X) M),
        X -> @mgcarrier M;
  Phi_nat_M : forall (X : Type) (T_X : IsHSet X)
              (M : MonoidalGroupoid) (G : MonoidalFunctor (F X T_X) M)
              (N : MonoidalGroupoid) (H : MonoidalFunctor M N),
              H o Phi X T_X M G == Phi X T_X N (MonoidalFunctor_comp H G);
  Psi : forall (X : Type) (T_X : IsHSet X)
        (M : MonoidalGroupoid) (g : X -> @mgcarrier M),
        MonoidalFunctor (F X T_X) M;
  Psi_nat_X : forall (X : Type) (T_X : IsHSet X)
              (M : MonoidalGroupoid) (g : X -> @mgcarrier M)
              (Y : Type) (T_Y : IsHSet Y) (h : Y -> X),
              MonoidalNatIso
                (MonoidalFunctor_comp
                  (Psi X T_X M g)
                  (@F_arr F free_functor Y X T_Y T_X h))
                (Psi Y T_Y M (g o h));
  Theta : forall (X : Type) (T_X : IsHSet X) (M : MonoidalGroupoid),
          Phi X T_X M o Psi X T_X M == idmap;
  Chi : forall (X : Type) (T_X : IsHSet X)
          (M : MonoidalGroupoid) (G : MonoidalFunctor (F X T_X) M),
          MonoidalNatIso (Psi X T_X M (Phi X T_X M G)) G
  }.
```

We now turn our attention to the monoidal groupoids we discussed in the thesis. The construction of lists is already available from the HoTT library; our formalization of the encode-decode proof that $\mathsf{list}(X)$ is a 0-type (`IsHSet_list`) for every 0-type X closely follows the one provided in Lemma 3.27, and we will not include it in this appendix.

The definition of the components of the monoidal structure of types of lists is also straightforward; here we display the definition of the monoidal groupoid with carrier $\mathsf{list}(X)$ in (3.35), in order to show the names we give to such components.

```
Definition listMG (X : Type) {T : IsHSet X} : MonoidalGroupoid
  := Build_MonoidalGroupoid (list X) (@trunc_succ 0 _ (IsHSet_list X))
     nil app alpha_list lambda_list rho_list pentagon_list triangle_list.
```

Above, the monoidal product app is the operation of list append $+\!\!+ : \mathsf{list}(X) \to \mathsf{list}(X) \to \mathsf{list}(X)$ in Definition 2.32, which is defined inductively on the first argument.

```
Fixpoint app {X : Type} (l : list X)
  : list X -> list X
  := fun k => match l with
    | nil => k
    | cons x l' => cons x (app l' k) end.
```

Given a type X, the the higher inductive type $\mathsf{FMG}(X)$ of free monoidal expressions (Definition 3.37) is implemented by specifying, in a private environment, its 0-constructors (on which Coq can perform pattern matching) and, separately, the higher constructors and the elimination principle of the HIT as axioms. This way of defining HITs follows the well-established scheme adopted in the HoTT library, which is due to Licata [Lic11].

```
Private Inductive FMG (X : Type) : Type :=
  | e : FMG X
  | iota : X -> FMG X
  | m : FMG X -> FMG X -> FMG X.
Global Arguments e {X}.
Global Arguments iota {X} _.
Global Arguments m {X} _ _.
Context (X : Type).

Definition mm {x x' y y' : FMG X}
  (p : x = x') (q : y = y')
  : m x y = m x' y'
  := ap011 m p q.

Axiom alpha : @IsAssociative (FMG X) m.
Axiom lambda : @IsLeftUnital (FMG X) e m.
Axiom rho : @IsRightUnital (FMG X) e m.
Axiom pentagon : IsPentagonCoherent m alpha.
Axiom triangle : IsTriangleCoherent e m alpha lambda rho.
Axiom T_FMG : IsTrunc 1 (FMG X).
```

The elimination principle $\mathsf{ind}_{\mathsf{FMG}}$ requires some attention. The type $\mathsf{FMG}(X)$ is *ap-recursive*; given a family $P : \mathsf{FMG}(X) \to \mathcal{U}$, the requirement corresponding to $\otimes$ in the definition of $\mathsf{ind}_{\mathsf{FMG}} : \Pi\,(x : \mathsf{FMG}(X))\,.\,P(x)$ is

$$\otimes' : \Pi\,(a, b : \mathsf{FMG}(X))\,.\,P(a) \to P(b) \to P(a \otimes b),$$

producing a term in the fiber of the product of two terms, given terms in the respective fibers. In order to define the requirements corresponding to $\pentagon$ and $\triangledown$, we need instances of the dependent application of $(- \otimes' -)$, i.e., a term

$$\begin{aligned}
[- \otimes' -]^{\mathsf{d}} : \Pi\,(a_1, a_2, b_1, b_2 : \mathsf{FMG}(X))\,.&\\
\Pi\left(a_1' : P(a_1),\ a_2' : P(a_2),\ b_1' : P(b_1),\ b_2' : P(b_2)\right)\,.&\\
\Pi\,(p : a_1 = a_2,\ q : b_1 = b_2)\,.&\\
\Pi\left(p' : p^P_*(a_1') =_{P(a_2)} a_2',\ q' : q^P_*(b_1') =_{P(b_2)} b_2'\right)\,.&\\
(p \otimes q)^P_*(a_1' \otimes' b_1') =_{P(a_2 \otimes b_2)} a_2' \otimes b_2',&
\end{aligned}$$

which can be obtained by induction on the paths p, q, p' and q'. In the excerpt below, this term appears as `mm'`.

```
Context
  (P : FMG X -> Type)
  (e' : P e)
  (iota' : forall x : X, P (iota x))
  (m' : forall (a b : FMG X), P a -> P b -> P (m a b)).
Global Arguments m' {a} {b} _ _.

Definition mm'
  {a1 a2 b1 b2 : FMG X} {p : a1 = a2} {q : b1 = b2}
  {a1' : P a1} {a2' : P a2} {b1' : P b1} {b2' : P b2}
  (p' : transport P p a1' = a2') (q' : transport P q b1' = b2')
  : transport P (mm p q) (m' a1' b1') = m' a2' b2'.
Proof.
  induction p, q, p', q'; constructor.
Defined.

Fixpoint FMG_ind
  (alpha' : forall (a b c : FMG X) (a' : P a) (b' : P b) (c' : P c),
    transport P (alpha a b c) (m' (m' a' b') c') = m' a' (m' b' c'))
  (lambda' : forall (b : FMG X) (b' : P b),
    transport P (lambda b) (m' e' b') = b')
  (rho' : forall (a : FMG X) (a' : P a),
    transport P (rho a) (m' a' e') = a')
  (pentagon' : forall (a b c d : FMG X) (a' : P a) (b' : P b) (c' : P c) (d' : P d),
    concat_D
      (alpha' _ _ _ (m' a' b') c' d')
      (alpha' _ _ _ a' b' (m' c' d'))
    = ap (fun z => transport P z (m' (m' (m' a' b') c') d')) (pentagon a b c d)
      @ concat_D
          (concat_D
            (mm' (alpha' _ _ _ a' b' c') (transport_1 P d'))
            (alpha' _ _ _ a' (m' b' c') d'))
          (mm' (transport_1 P a') (alpha' _ _ _ b' c' d')))
  (triangle' : forall (a b : FMG X) (a' : P a) (b' : P b),
    concat_D
      (alpha' _ _ _ a' e' b')
      (mm' (transport_1 P a') (lambda' _ b'))
    = ap (fun z => transport P z (m' (m' a' e') b')) (triangle a b)
      @ mm' (rho' _ a') (transport_1 P b'))
  (T' : forall (w : FMG X), IsTrunc 1 (P w))
  (w : FMG X) : P w
  := match w with
    | e => e'
    | iota x => iota' x
    | m a b =>
        m'
          (FMG_ind alpha' lambda' rho' pentagon' triangle' T' a)
          (FMG_ind alpha' lambda' rho' pentagon' triangle' T' b) end.
```

Above, `concat_D` is $(- \cdot^{\mathsf{d}} -)$ (from Definition 2.60(vii)), while `transport_1` proves $\mathsf{refl}^P_*(u) = u$ (the identity holds judgmentally).

We see that `FMG_ind` computes on terms of the form e, $\iota(x)$ and $a \otimes b$, while for α, λ and ρ we need axioms along the lines of:

```
Axiom FMG_ind_beta_lambda
```

```
  : forall (b : FMG X),
    apD (FMG_ind P e' iota' m' alpha' lambda' rho' pentagon' triangle' T')
      (lambda b)
    = lambda' b (FMG_ind b).
```

The computation rules relative to $\pentagon$ and $\triangle$ were not formalized, since we never use them (as mentioned in Remark 2.134).

The non-dependent version $\mathsf{rec}_{\mathsf{FMG}}$ of the elimination principle can be derived from $\mathsf{ind}_{\mathsf{FMG}}$. We omit the construction here, as it only consists of tedious path algebra. Other specific (derived) versions of the elimination principle were also formalized, applying to cases when we eliminate into (families of) 0- or (-1)-types or, even more specifically, to families of paths in 1- or 0-types.

$\mathsf{FMG}(X)$ is a monoidal groupoid:

```
Definition FMG_MG (X : Type) {T_X : IsHSet X} : MonoidalGroupoid
  := Build_MonoidalGroupoid (FMG X) T_FMG e m alpha lambda rho pentagon triangle.
```

Functoriality of FMG and the proof of freeness are described in detail in Chapter 3. Here we only include the function underlying the monoidal functor obtained by functoriality from a function between types (Lemma 3.38):

```
Definition FMG_functor_fun {X Y : Type} {T_X : IsHSet X} {T_Y : IsHSet Y}
  (h : X -> Y)
  : FMG X -> FMG Y
  := FMG_rec X (FMG Y) e (iota o h) m alpha lambda rho pentagon triangle T_FMG.
```

where `FMG_rec` is $\mathsf{rec}_{\mathsf{FMG}}$ and `e`, `iota`, ... are the constructors of $\mathsf{FMG}(Y)$.

The definitions of the monoidal functors K and J and of the monoidal natural isomorphism η follow the ones given in Chapter 3; we present here the underlying functions of K and J (respectively, Definition 3.49 and Definition 3.50).

```
Definition K_fun
  : FMG X -> list X
  := FMG_rec X (list X) nil (fun x => cons x nil) app alpha_list lambda_list rho_list
     pentagon_list triangle_list (@trunc_succ 0 (list X) (IsHSet_list X)).

Fixpoint J_fun (l : list X)
  : FMG X
  := match l with
    | nil => e
    | x :: l => m (iota x) (J_fun l) end.
```

The proof of coherence (Theorem 3.26) uses path algebra, the underlying homotopy in η and the proof that the type of lists over a 0-type is a 0-type.

```
Theorem FMG_coherence
  : forall {a b : FMG X} (p q : a = b), p = q.
Proof.
  intros.
  refine ((moveR_pV _ _ _ (concat_pA1 eta_homotopy p))^ @ _
    @ moveR_pV _ _ _ (concat_pA1 eta_homotopy q)).
  apply whiskerR; apply whiskerL.
  refine (ap_compose K J p @ _ @ (ap_compose K J q)^).
  apply ap.
  srapply @set_list.
Defined.

Corollary IsHSet_FMG
  : IsHSet (FMG X).
Proof.
  srapply hset_axiomK.
  exact (fun x p => FMG_coherence p idpath).
Defined.
```

A.2 Coherence for Symmetric Monoidal Groupoids

The formalization of symmetric monoidal groupoid follows closely the one of monoidal groupoids. In particular, we define new classes for symmetric monoidal groupoids, functors and natural isomorphisms; these could in principle be built upon the definition given in Appendix A.1, by stating that a symmetric monoidal groupoid is a monoidal groupoid with additional data. However, we encountered some issues as Coq failed to recognize the components of the newly defined class properly. For this reason, we decided to define specific classes anew, even though it resulted in code duplication.

The code relevant to Definition 4.1 is as follows:

```
Definition IsSymmetric : Type
  := forall a b : X, m a b = m b a.

Definition IsHexagonCoherent (alpha : IsAssociative) (tau : IsSymmetric) : Type
  := forall a b c : X,
      alpha a b c @ tau a (m b c) @ alpha b c a
      = ap011 m (tau a b) (idpath c) @ alpha b a c @ ap011 m (idpath b) (tau a c).

Definition IsBigonCoherent (tau : IsSymmetric) : Type
  := forall a b : X,
      tau a b @ tau b a = idpath (m a b).
```

```
Class SymMonoidalGroupoid := {
  smgcarrier : Type;
  smgtrunc : IsTrunc 1 smgcarrier;
  smg_e : smgcarrier;
  smg_m : smgcarrier -> smgcarrier -> smgcarrier;
```

```
  smg_alpha : IsAssociative smg_m;
  smg_lambda : IsLeftUnital smg_e smg_m;
  smg_rho : IsRightUnital smg_e smg_m;
  smg_tau : IsSymmetric smg_m;
  smg_pentagon : IsPentagonCoherent smg_m smg_alpha;
  smg_triangle : IsTriangleCoherent smg_e smg_m smg_alpha smg_lambda smg_rho;
  smg_hexagon : IsHexagonCoherent smg_m smg_alpha smg_tau;
  smg_bigon : IsBigonCoherent smg_m smg_tau
  }.
```

The HIT of symmetric lists (Definition 4.11) is implemented as below:

```
Private Inductive slist (X : Type) : Type :=
  | nil : slist X
  | cons : X -> slist X -> slist X.
Global Arguments nil {X}.
Global Arguments cons {X} _ _.

End slist_private.

Declare Scope slist_scope.
Infix "::" := cons (at level 60, right associativity) : slist_scope.
Open Scope slist_scope.

Section slist.

Context {X : Type}.

Axiom swap
  : forall (x y : X) (l : slist X),
    x :: y :: l = y :: x :: l.

Axiom double
  : forall (x y : X) (l : slist X),
    swap x y l @ swap y x l = idpath.

Axiom triple
  : forall (x y z : X) (l : slist X),
  swap x y (cons z l) @ ap (cons y) (swap x z l) @ swap y z (cons x l)
  = ap (cons x) (swap y z l) @ swap x z (cons y l) @ ap (cons z) (swap x y l).

Axiom T_slist
  : IsTrunc 1 (slist X).

Section slist_ind.

Definition swap'_ind_type
  (P : slist X -> Type)
  (cons' : forall (x : X) (l : slist X) (IHl : P l), P (x :: l))
  : Type
  := forall (x y : X) (l : slist X) (IHl : P l),
    transport P (swap x y l) (cons' x _ (cons' y _ IHl)) = cons' y _ (cons' x _ IHl).

Definition double'_ind_type
  (P : slist X -> Type)
  (cons' : forall (x : X) (l : slist X) (IHl : P l), P (x :: l))
  (swap' : swap'_ind_type P cons')
  : Type
```

```
  := forall (x y : X) (l : slist X) (IHl : P l),
    concat_D (swap' x y l IHl) (swap' y x l IHl)
    = ap (fun z => transport P z (cons' x _ (cons' y _ IHl))) (double x y l).

Definition triple'_ind_type
  (P : slist X -> Type)
  (cons' : forall (x : X) (l : slist X) (IHl : P l), P (x :: l))
  (swap' : swap'_ind_type P cons')
  : Type
  := forall (x y z : X) (l : slist X) (IHl : P l),
    concat_D
      (concat_D
        (swap' x y _ (cons' z _ IHl))
        (transport_ap P (cons y) (cons' y) _ (swap' x z _ IHl)))
      (swap' y z _ (cons' x _ IHl))
    = ap
        (fun w => transport P w (cons' x _ (cons' y _ (cons' z _ IHl))))
        (triple x y z l)
      @ concat_D
          (concat_D
            (transport_ap P (cons x) (cons' x) _ (swap' y z _ IHl))
            (swap' x z _ (cons' y _ IHl)))
          (transport_ap P (cons z) (cons' z) _ (swap' x y _ IHl)).

Fixpoint slist_ind
  (P : slist X -> Type)
  (nil' : P nil)
  (cons' : forall (x : X) (l : slist X) (IHl : P l), P (x :: l))
  (swap' : swap'_ind_type P cons')
  (double' : double'_ind_type P cons' swap')
  (triple' : triple'_ind_type P cons' swap')
  (T_slist' : forall (l : slist X), IsTrunc 1 (P l))
  (l : slist X)
  : P l
  := match l with
    | nil => nil'
    | x :: k => cons' x k (slist_ind P nil' cons' swap' double' triple' T_slist' k)
    end.

Axiom slist_ind_beta_swap
  : forall (P : slist X -> Type)
  (nil' : P nil)
  (cons' : forall (x : X) (l : slist X) (IHl : P l), P (x :: l))
  (swap' : swap'_ind_type P cons')
  (double' : double'_ind_type P cons' swap')
  (triple' : triple'_ind_type P cons' swap')
  (T_slist' : forall (l : slist X), IsTrunc 1 (P l))
  (x y : X) (l : slist X),
  apD (slist_ind P nil' cons' swap' double' triple' T_slist') (swap x y l)
  = swap' x y _ (slist_ind P nil' cons' swap' double' triple' T_slist' l).

End slist_ind.
```

The non-dependent version `slist_rec` of the elimination principle is derived.

The definition of symmetric list append ($+\!\!+ : \mathsf{slist}(X) \to \mathsf{slist}(X) \to \mathsf{slist}(X)$ in Definition 4.13) by means of the elimination principle of $\mathsf{slist}(X)$ requires function extensionality: indeed, it behaves like list append on 0-constructors, but in order

to verify that the definition respects the 1-constructor swap we need to provide an identity between functions. Below, `path_forall` is fxt, while `path_forall_1` and `path_forall_pp` correspond to the laws in Lemma 2.117.

```
Context `{Funext}.
Context {X : Type}.

  Lemma sapp_cons
    : X -> (slist X -> slist X) -> slist X -> slist X.
  Proof.
    intros x f. exact (cons x o f).
  Defined.

  Lemma sapp_swap
    : swap'_rec_type (slist X -> slist X) sapp_cons.
  Proof.
    unfold_slist_types.
    intros x y f. unfold sapp_cons.
    exact (path_forall _ _ (fun l => swap x y (f l))).
  Defined.

  Lemma sapp_double
    : double'_rec_type (slist X -> slist X) sapp_cons sapp_swap.
  Proof.
    unfold_slist_types.
    intros x y f.
    unfold sapp_swap.
    refine ((path_forall_pp _ _ _ (fun l => swap x y (f l)) (fun l => swap y x (f l)))^
      @ _).
    refine (_ @ path_forall_1 (fun l : slist X => x :: y :: f l)).
    apply ap. srapply @path_forall; intro l.
    apply double.
  Qed.

  Lemma sapp_triple
    : triple'_rec_type (slist X -> slist X) sapp_cons sapp_swap.
  Proof.
    unfold_slist_types.
    intros x y z f.
    rewrite (ap_pf_2 (swap x z)), (ap_pf_2 (swap y z)), (ap_pf_2 (swap x y)).
    unfold sapp_swap, sapp_cons.
    rewrite <- path_forall_pp, <- path_forall_pp, <- path_forall_pp, <- path_forall_pp.
    apply ap. apply path_forall; intro l.
    apply triple.
  Qed.

  Definition sapp_trunc
    : IsTrunc 1 (slist X -> slist X).
  Proof.
    exact (@trunc_forall H (slist X) (fun _ => slist X) 1 (fun _ => T_slist)).
  Defined.

Definition sapp
  : slist X -> slist X -> slist X
  := slist_rec _ idmap sapp_cons sapp_swap sapp_double sapp_triple sapp_trunc.
```

The definition of the HIT $\mathsf{FSMG}(X)$ and the proof of freeness of the construction

follow the ones given for $\mathsf{FMG}(X)$. The construction of a symmetric monoidal structure on $\mathsf{list}(X)$ and the proof of coherence for symmetric monoidal groupoid entail a number of application of the elimination principles of higher inductive types and are omitted here, for they are described in detail in Chapter 4.

A.3 Finite Types and Symmetric Monoidal Structures

The subuniverse $\mathcal{B}\mathsf{S}_n$ of finite types of cardinality n (Definition 5.3) is implemented as a Σ-type:

```
Definition BS : nat -> Type
  := fun n => {A : Type & merely (A <~> Fin n)}.

Definition BS_dot
  : Type
  := sig BS.
```

The proof that $\mathcal{B}\mathsf{S}_n$ is a 1-type for every $n : \mathbb{N}$ (Corollary 5.7) is included below:

```
Lemma BS_trunc (n : nat)
  : IsTrunc 1 (BS n).
Proof.
  unfold BS. srapply @trunc_sigma';
  change (forall X Y : BS n, IsHSet (X.1 = Y.1)).
  intros [A tA] [B tB]; simpl.
  srapply @istrunc_paths_Type.
  apply hset_Finite.
  exact (Build_Finite B n tB).
Defined.
```

where `trunc_sigma'` (Lemma 2.79) is defined as follows:

```
Definition trunc_sigma'
  {A : Type} {B : A -> Type} {n : trunc_index}
  {TB : forall a : A, IsTrunc n.+1 (B a)}
  {Ts : forall p q : sig B, IsTrunc n (p.1 = q.1)}
  : IsTrunc n.+1 (sig B).
Proof.
  intros x y.
  enough (Tp : (IsTrunc n {p : x.1 = y.1 & transport B p x.2 = y.2})).
  { revert Tp. srapply @trunc_equiv'.
  exact (Build_Equiv _ _ (path_sigma_uncurried B x y) _). }
  srapply @trunc_sigma.
Defined.
```

$\mathcal{B}\mathsf{S}_0$ and $\mathcal{B}\mathsf{S}_1$ are contractible (Lemma 5.8); here we show the former.

```
Lemma Contr_BS0
  : Contr (BS 0).
```

```
Proof.
  unfold BS. srapply @Build_Contr.
  + exact (Fin 0; tr equiv_idmap).
  + intros [A t].
    srapply @sigma_truncfib. symmetry.
    strip_truncations.
    exact (path_universe_uncurried t).
Defined.
```

Above, `sigma_truncfib` is the function term described in Remark 2.78.

$\mathcal{B}\mathrm{S}_n$ is connected for every $n : \mathbb{N}$ (Lemma 5.9):

```
Lemma Connected_BS
  : forall n : nat, Contr (Trunc 0 (BS n)).
Proof.
  intro n. srapply @Build_Contr.
  + apply tr. exact (Fin n; tr equiv_idmap).
  + srapply @Trunc_ind.
    intros [A t]. strip_truncations.
    apply ap. srapply @sigma_truncfib.
    exact (path_universe_uncurried t)^.
Defined.
```

Following Lemma 5.11, we define a symmetric monoidal structure on the type

```
Definition BS_dot'
  : Type
  := {X : nat * Type & merely (snd X <~> Fin (fst X))}.
```

The symmetric monoidal product is:

```
Definition BS_dot'_product
  : BS_dot' -> BS_dot' -> BS_dot'.
Proof.
  unfold BS_dot', BS.
  srapply m_base_fiber.
  + intros [nA A] [nB B]. exact (nA + nB, (A + B)).
  + intros ?? tA tB; simpl in *. exact (fin_coproduct tA tB).
Defined.
```

where we use:

```
Lemma fin_coproduct {nA nB : nat} {A B : Type}
  (tA : Trunc (-1) (A <~> Fin nA)) (tB : Trunc (-1) (B <~> Fin nB))
  : Trunc (-1) (A + B <~> Fin (nA + nB)).

...

Lemma m_base_fiber
  (m_base : A -> A' -> A'')
  (m_fiber : forall (a : A) (a' : A'), B a -> B' a' -> B'' (m_base a a'))
  : sig B -> sig B' -> sig B''.
```

A symmetric monoidal structure on $\mathbb{N}$ is easily found; for a symmetric monoidal structure on $\mathcal{U}$ we use constructions such as:

```
Lemma alpha_U_equiv
  : forall A B C : Type, ((A + B) + C) <~> (A + (B + C)).
Proof.
  srapply equiv_sum_assoc.
Defined.

Lemma alpha_U
  : forall A B C : Type, ((A + B) + C) = (A + (B + C)).
Proof.
  intros; srapply @path_universe_uncurried. srapply alpha_U_equiv.
Defined.

Lemma pentagon_U_equiv
  : forall A B C D : Type,
    alpha_U_equiv A B (C + D) oE alpha_U_equiv (A + B) C D
    = (1 +E alpha_U_equiv B C D)
      oE (alpha_U_equiv A (B + C) D oE (alpha_U_equiv A B C +E 1)).
Proof.
  intros. apply path_equiv. apply path_forall.
  intros [[[a|b]|c]|d]; constructor.
Defined.

Lemma pentagon_U
  : IsPentagonCoherent (fun A B => A + B) alpha_U.

...
```

We then combine those structures to obtain one for `BS_dot'`:

```
Lemma alpha_BS_dot'
  : forall a b c : BS_dot',
    BS_dot'_product (BS_dot'_product a b) c = BS_dot'_product a (BS_dot'_product b c).
Proof.
  unfold BS_dot', BS, BS_dot'_product.
  intros [[nA A] tA] [[nB B] tB] [[nC C] tC].
  srapply @sigma_truncfib. srapply @path_prod'.
  + apply alpha_nat.
  + apply alpha_U.
Defined.
```

We omit the rest of the formalization here.

The indexed family $\mathsf{del} : \mathbb{N} \to \mathcal{U}$ of HITs representing the deloopings of symmetric groups (Definition 5.18) is implemented as follows.

```
Section deloop.
Open Scope nat.

Private Inductive deloop : nat -> Type :=
  | dlpt0 : deloop 0
  | dli : forall n : nat, deloop n -> deloop n.+1.
Global Arguments dli {n} _.
```

```
Fixpoint dlpt (n : nat) : deloop n
  := match n with
  | 0 => dlpt0
  | n'.+1 => dli (dlpt n') end.

Axiom dltw : forall (n : nat) (a : deloop n),
  dli (dli a) = dli (dli a)
  :> deloop n.+2.
Global Arguments dltw {n} _.

Axiom dldo : forall (n : nat) (a : deloop n),
  dltw a @ dltw a = idpath
  :> (dli (dli a) = dli (dli a)).
Global Arguments dldo {n} _.

  Definition dldo_ : forall (n : nat) (a : deloop n),
    (dltw a)^ = dltw a.
  Proof.
    intros n a.
    refine (cancelR (dltw a)^ (dltw a) (dltw a) _).
    exact (concat_Vp _ @ (dldo a)^).
  Defined.
  Global Arguments dldo_ {n} _.

Axiom dlbr : forall (n : nat) (a : deloop n),
  dltw (dli a) @ ap dli (dltw a) @ dltw (dli a)
  = ap dli (dltw a) @ dltw (dli a) @ ap dli (dltw a)
  :> (dli (dli (dli a)) = dli (dli (dli a))).
Global Arguments dlbr {n} _.

Axiom dlT : forall (n : nat), IsTrunc 1 (deloop n).

Definition deloop_dot
  := sig deloop.

Definition dlT_dot
  : IsTrunc 1 deloop_dot.
Proof.
  srapply @trunc_sigma; intro; apply dlT.
Defined.

Definition dlpt0_dot
  : deloop_dot
  := (0; dlpt0).

Definition Sdli
  : deloop_dot -> deloop_dot
  := sigma_function S (@dli).

Section deloop_ind.

Definition dltw'_type
  (P : forall n : nat, deloop n -> Type)
  (dli' : forall (n : nat) (a : deloop n) (a' : P n a), P n.+1 (dli a))
  : Type
  := forall (n : nat) (a : deloop n) (a' : P n a),
  transport (P n.+2) (dltw a) (dli' _ _ (dli' _ _ a')) = dli' _ _ (dli' _ _ a').

Definition dldo'_type
```

```
  (P : forall n : nat, deloop n -> Type)
  (dli' : forall (n : nat) (a : deloop n) (a' : P n a), P n.+1 (dli a))
  (dltw' : dltw'_type P dli')
  : Type
  := forall (n : nat) (a : deloop n) (a' : P n a),
  (ap (fun z => transport (P n.+2) z (dli' _ _ (dli' _ _ a'))) (dldo a))^
  @ concat_D (dltw' _ _ a') (dltw' _ _ a')
  = idpath.

Definition dlbr'_type
  (P : forall n : nat, deloop n -> Type)
  (dli' : forall (n : nat) (a : deloop n) (a' : P n a), P n.+1 (dli a))
  (dltw' : dltw'_type P dli')
  : Type
  := forall (n : nat) (a : deloop n) (a' : P n a),
  (ap (fun z => transport (P n.+3) z (dli' _ _ (dli' _ _ (dli' _ _ a')))) (dlbr a))^ @
  concat_D
    (concat_D
      (dltw' _ _ (dli' _ _ a'))
      (transport_ap_nat (@dli) (@dli') (dltw a) (dltw' _ _ a')))
    (dltw' _ _ (dli' _ _ a'))
  = concat_D
    (concat_D
      (transport_ap_nat (@dli) (@dli') (dltw a) (dltw' _ _ a'))
      (dltw' _ _ (dli' _ _ a')))
    (transport_ap_nat (@dli) (@dli') (dltw a) (dltw' _ _ a')).

Context (P : forall n : nat, deloop n -> Type)
  (dlpt0' : P 0 dlpt0)
  (dli' : forall (n : nat) (a : deloop n) (a' : P n a), P n.+1 (dli a)).
Arguments dli' {n} {a} _.

Fixpoint deloop_ind
  (dltw' : dltw'_type P (@dli'))
  (dldo' : dldo'_type P (@dli') dltw')
  (dlbr' : dlbr'_type P (@dli') dltw')
  (dlT' : forall (n : nat) (x : deloop n), IsTrunc 1 (P n x))
  (n : nat) (s : deloop n)
  : P n s
  := match s with
  | dlpt0 => fun _ => fun _ => fun _ => fun _ => dlpt0'
  | dli n s0 => fun _ => fun _ => fun _ => fun _ =>
    dli' (deloop_ind dltw' dldo' dlbr' dlT' n s0)
  end dltw' dldo' dlbr' dlT'.

Axiom deloop_ind_beta_dltw
  : forall
  (dltw' : dltw'_type P (@dli'))
  (dldo' : dldo'_type P (@dli') dltw')
  (dlbr' : dlbr'_type P (@dli') dltw')
  (dlT' : forall (n : nat) (x : deloop n), IsTrunc 1 (P n x))
  (n : nat) (a : deloop n),
  apD (deloop_ind dltw' dldo' dlbr' dlT' n.+2) (dltw a) =
  dltw' _ _ (deloop_ind dltw' dldo' dlbr' dlT' n a).

Lemma deloop_ind_to_set
  (dltw' : dltw'_type P (@dli'))
  (dlT' : forall (n : nat) (x : deloop n), IsTrunc 0 (P n x))
  (n : nat) (s : deloop n)
  : P n s.
```

```
Proof.
  srapply @deloop_ind.
  { exact dltw'. }
  all: repeat intro; srapply @path_ishprop.
Defined.

Lemma deloop_ind_to_prop
  (dlT' : forall (n : nat) (x : deloop n), IsHProp (P n x))
  (n : nat) (s : deloop n)
  : P n s.
Proof.
  srapply @deloop_ind_to_set.
  repeat intro; srapply @path_ishprop.
Defined.

End deloop_ind.
```

Because $\mathsf{del}_{(-)}$ is both indexed and ap-recursive, in the specification of the elimination principle above (specifically, in the requirement for br) we make use of the term `transport_ap_nat`, which is defined as follows:

```
Lemma transport_ap_nat
  {A : nat -> Type} {B : forall n : nat, A n -> Type}
  (f : forall n : nat, A n -> A n.+1)
  (g : forall (n : nat) (a : A n), B n a -> B n.+1 (f n a))
  {n : nat} {x y : A n} (p : x = y) {x' : B n x} {y' : B n y}
  (h : transport (B n) p x' = y')
  : transport (B n.+1) (ap (f n) p) (g n x x') = g n y y'.
Proof.
  refine (_ @ ap (g n y) h).
  induction p. constructor.
Defined.
```

The non-dependent elimination principle `deloop_rec` is derived accordingly. Following Lemma 5.37, we can further specialize the non-dependent elimination principle for when we eliminate into Σ-types whose family is a family of (-1)-types (such as subuniverses):

```
Lemma deloop_rec_truncfib
  (A : Type)
  (Q : nat -> A -> Type)
  (T : forall n : nat, IsTrunc 1 {a : A & Q n a})
  (Tf : forall (n : nat) (a : A), IsHProp (Q n a))
  (dlpt0' : A)
  (dlpt0'q : Q 0 dlpt0')
  (dli' : nat -> A -> A)
  (dli'q : forall (n : nat) (a : A), Q n a -> Q n.+1 (dli' n a))
  (dltw' : forall (n : nat) (a : A), dli' n.+1 (dli' n a) = dli' n.+1 (dli' n a))
  (dldo' : forall (n : nat) (a : A), dltw' n a @ dltw' n a = idpath)
  (dlbr' : forall (n : nat) (a : A),
    (dltw' n.+1 (dli' n a) @ ap (dli' n.+2) (dltw' n a)) @ dltw' n.+1 (dli' n a)
     = (ap (dli' n.+2) (dltw' n a) @ dltw' n.+1 (dli' n a)) @ ap (dli' n.+2) (dltw' n a))
  : forall (n : nat), deloop n -> {a : A & Q n a}.
Proof.
```

```
  srapply @deloop_rec; hnf.
  + exists dlpt0'. exact dlpt0'q.
  + intro n. exact (sigma_function (dli' n) (dli'q n)).
  + simpl. intros n [a q]; unfold sigma_function.
    srapply @sigma_truncfib. apply dltw'.
  + abstract (intros n [a q];
    refine (sigma_truncfib_concat _ _ (dltw' n a) (dltw' n a) @ _
      @ @sigma_truncfib_1 _ (fun a : A => Q n.+2 a) _ _ (Tf n.+2) _ idpath);
    apply ap; apply dldo').
  + abstract (intros n [a q]; simpl;
    rewrite ap_sigma_function_p;
    unfold path_sigma_function;
    rewrite path_path_sigma'_concat, path_path_sigma'_concat,
      path_path_sigma'_concat, path_path_sigma'_concat;
    srapply @path_path_sigma'_truncfib;
    rewrite sigma_truncfib_pr1;
    apply dlbr').
Defined.

Lemma deloop_rec_truncfib_beta_dltw
  (A : Type)
  (Q : nat -> A -> Type)
  (T : forall n : nat, IsTrunc 1 {a : A & Q n a})
  (Tf : forall (n : nat) (a : A), IsHProp (Q n a))
  (dlpt0' : A)
  (dlpt0'q : Q 0 dlpt0')
  (dli' : nat -> A -> A)
  (dli'q : forall (n : nat) (a : A), Q n a -> Q n.+1 (dli' n a))
  (dltw' : forall (n : nat) (a : A), dli' n.+1 (dli' n a) = dli' n.+1 (dli' n a))
  (dldo' : forall (n : nat) (a : A), dltw' n a @ dltw' n a = idpath)
  (dlbr' : forall (n : nat) (a : A),
    (dltw' n.+1 (dli' n a) @ ap (dli' n.+2) (dltw' n a)) @ dltw' n.+1 (dli' n a)
    = (ap (dli' n.+2) (dltw' n a) @ dltw' n.+1 (dli' n a)) @ ap (dli' n.+2) (dltw' n a))
  (n : nat) (a : deloop n)
  : ap pr1
      (ap
        (deloop_rec_truncfib A Q T Tf dlpt0' dlpt0'q dli' dli'q dltw' dldo' dlbr' n.+2)
        (dltw a))
    = dltw' n
        (deloop_rec_truncfib A Q T Tf dlpt0' dlpt0'q dli' dli'q dltw' dldo' dlbr' n a).1.
Proof.
  unfold deloop_rec_truncfib.
  refine (ap (ap pr1) (deloop_rec_beta_dltw _ _ _ _ _ _ _ _ a) @ _).
  apply sigma_truncfib_pr1.
Defined.

Definition deloop_rec_subuniverse
  := deloop_rec_truncfib Type.

Definition deloop_rec_subuniverse_beta_dltw
  := deloop_rec_truncfib_beta_dltw Type.
```

The types del_0 and del_1 are contractible (Lemma 5.25); here we show it for del_0:

```
Lemma deloop_contr_0 `{Funext}
  : forall (n : nat) (a : deloop n), n = 0 -> dlpt n = a.
Proof.
  srapply @deloop_ind_to_set; simpl.
```

```
  + intros; exact idpath.
  + intros n b IH p. apply Empty_rec. exact (not_sn_0 p).
  + unfold dltw'_type. intros. srapply @path_forall. intro p.
    apply Empty_rec. exact (not_sn_0 p).
  + intros; srapply @trunc_forall; intros; srapply @dlT.
Defined.

Lemma Contr_deloop0 `{Funext}
  : Contr (deloop 0).
Proof.
  srapply @Build_Contr.
  + apply dlpt.
  + intro a. exact (deloop_contr_0 0 a idpath).
Defined.
```

The type del_n is connected for every $n : \mathbb{N}$ (Lemma 5.26):

```
Lemma Connected_deloopn
  : forall n : nat, Contr (Trunc 0 (deloop n)).
Proof.
  intro n. srapply @Build_Contr.
  + apply tr. exact (dlpt n).
  + srapply @Trunc_ind.
    revert n. srapply @deloop_ind_to_prop; simpl.
    - constructor.
    - intros m b p.
      exact (ap (truncmap 0 dli) p).
Defined.
```

As we saw in Lemma 5.30, the type $\mathsf{del}_\bullet$ has a symmetric monoidal structure. Here we show the implementation of the symmetric monoidal product $\oplus : \mathsf{del}_\bullet \to \mathsf{del}_\bullet \to \mathsf{del}_\bullet$ (Definition 5.28):

```
  Definition deloop_m_dli
    : (deloop_dot -> deloop_dot) -> deloop_dot -> deloop_dot.
  Proof.
    intro a'. exact (Sdli o a').
  Defined.

  Definition deloop_m_dltw
    : forall a' : deloop_dot -> deloop_dot,
      deloop_m_dli (deloop_m_dli a') = deloop_m_dli (deloop_m_dli a').
  Proof.
    unfold deloop_m_dli.
    intros. srapply @path_forall; intro x.
    apply dltw_dot.
  Defined.

  Definition deloop_m_dldo
    : forall a' : deloop_dot -> deloop_dot, deloop_m_dltw a' @ deloop_m_dltw a' = 1%path.
  Proof.
    unfold deloop_m_dltw.
    intros.
    refine ((path_forall_pp _ _ _ _ _)^ @ _).
    refine (_ @ path_forall_1 _).
    apply ap. srapply @path_forall; intro x.
```

```
    apply dldo_dot.
  Defined.

  Definition deloop_m_dlbr
    : forall a' : deloop_dot -> deloop_dot,
      (deloop_m_dltw (deloop_m_dli a') @ ap deloop_m_dli (deloop_m_dltw a'))
      @ deloop_m_dltw (deloop_m_dli a')
      = (ap deloop_m_dli (deloop_m_dltw a') @ deloop_m_dltw (deloop_m_dli a'))
        @ ap deloop_m_dli (deloop_m_dltw a').
  Proof.
    unfold deloop_m_dltw, deloop_m_dli.
    intros.
    rewrite ap_compose_pf.
    repeat rewrite <- path_forall_pp.
    apply ap. srapply @path_forall; intro x.
    apply dlbr_dot.
  Defined.

Definition deloop_m
  : deloop_dot -> deloop_dot -> deloop_dot.
Proof.
  srapply @sig_rect; intro n; revert n; hnf.
  assert (T : (IsTrunc 1 (deloop_dot -> deloop_dot))).
  { exact (@trunc_forall _ _ _ _ (fun _ => dlT_dot)). }
  srefine (deloop_rec_const (deloop_dot -> deloop_dot) _ _ _ _ _ _); clear T.
  + exact idmap.
  + exact deloop_m_dli.
  + exact deloop_m_dltw.
  + exact deloop_m_dldo.
  + exact deloop_m_dlbr.
Defined.
```

The proof of the symmetric monoidal equivalence $\mathsf{slist}(\mathbf{1}) \simeq \mathsf{del}_\bullet$ follows the one given in Theorem 5.35. Here we include the definitions of the functions $\ell : \mathsf{slist}(\mathbf{1}) \to \mathsf{del}_\bullet$ and $\jmath : \mathsf{del}_\bullet \to \mathsf{slist}(\mathbf{1})$ (Definition 5.32 and Definition 5.34 respectively).

```
  Lemma cons'_for_slist_Unit_to_deloop
    : Unit -> deloop_dot -> deloop_dot.
  Proof.
    intros _. exact Sdli.
  Defined.

  Lemma swap'_for_slist_Unit_to_deloop
    : swap'_rec_type deloop_dot cons'_for_slist_Unit_to_deloop.
  Proof.
    unfold swap'_rec_type.
    intros [] [] a; unfold cons'_for_slist_Unit_to_deloop.
    exact (dltw_dot a).
  Defined.

  Lemma double'_for_slist_Unit_to_deloop
    : double'_rec_type
      deloop_dot cons'_for_slist_Unit_to_deloop swap'_for_slist_Unit_to_deloop.
  Proof.
    unfold double'_rec_type.
    intros [] [] a; unfold swap'_for_slist_Unit_to_deloop.
    exact (dldo_dot a).
```

```
  Qed.

  Lemma triple'_for_slist_Unit_to_deloop
    : triple'_rec_type
      deloop_dot cons'_for_slist_Unit_to_deloop swap'_for_slist_Unit_to_deloop.
  Proof.
    unfold triple'_rec_type.
    intros [] [] [] a;
    unfold cons'_for_slist_Unit_to_deloop, swap'_for_slist_Unit_to_deloop.
    exact (dlbr_dot a).
  Qed.

Definition slist_Unit_to_deloop
  : slist Unit -> deloop_dot.
Proof.
  srapply @slist_rec.
  + exact (0; dlpt0).
  + exact cons'_for_slist_Unit_to_deloop.
  + exact swap'_for_slist_Unit_to_deloop.
  + exact double'_for_slist_Unit_to_deloop.
  + exact triple'_for_slist_Unit_to_deloop.
  + exact dlT_dot.
Defined.

  Lemma dli'_for_deloop_to_slist_Unit
    : nat -> slist Unit -> slist Unit.
  Proof.
    intros _ l; exact (cons tt l).
  Defined.

  Lemma dltw'_for_deloop_to_slist_Unit
    : dltw'_rec_type (fun _ : nat => slist Unit) dli'_for_deloop_to_slist_Unit.
  Proof.
    unfold dltw'_rec_type, dli'_for_deloop_to_slist_Unit.
    intros _ l. exact (swap tt tt l).
  Defined.

  Lemma dldo'_for_deloop_to_slist_Unit
    : dldo'_rec_type (fun _ : nat => slist Unit)
        dli'_for_deloop_to_slist_Unit dltw'_for_deloop_to_slist_Unit.
  Proof.
    unfold dldo'_rec_type, dltw'_for_deloop_to_slist_Unit.
    intros ? l; exact (double tt tt l).
  Defined.

  Lemma dlbr'_for_deloop_to_slist_Unit
    : dlbr'_rec_type (fun _ : nat => slist Unit)
        dli'_for_deloop_to_slist_Unit dltw'_for_deloop_to_slist_Unit.
  Proof.
    unfold dlbr'_rec_type, dltw'_for_deloop_to_slist_Unit, dli'_for_deloop_to_slist_Unit.
    intros _ l.
    change ((swap tt tt (cons tt l) @ ap (cons tt) (swap tt tt l))
      @ swap tt tt (cons tt l)
      = (ap (cons tt) (swap tt tt l) @ swap tt tt (cons tt l))
      @ ap (cons tt) (swap tt tt l)).
    exact (triple tt tt tt l).
  Defined.

Definition deloop_to_slist_Unit
  : deloop_dot -> slist Unit.
```

```
Proof.
  intros [n a]; revert a; revert n.
  srapply @deloop_rec; simpl.
  + exact nil.
  + exact dli'_for_deloop_to_slist_Unit.
  + exact dltw'_for_deloop_to_slist_Unit.
  + exact dldo'_for_deloop_to_slist_Unit.
  + exact dlbr'_for_deloop_to_slist_Unit.
  + intro; apply T_slist.
Defined.
```

The families of functions $\mathfrak{f} : \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \to \mathcal{BS}_n$ and $\mathfrak{f}^{\mathsf{b}} : \Pi\,(n : \mathbb{N})\,.\,\mathsf{del}_n \to \mathcal{U}$ in Definition 5.39 are given as follows:

```
  Definition dtb_dlpt0b
    : Type
    := Fin 0.

  Definition dtb_dlpt0f
    : Trunc (-1) (dtb_dlpt0b <~> Fin 0)
    := (tr 1%equiv).

  Definition dtb_dlib
    : nat -> Type -> Type
    := fun _ => add.

  Definition dtb_dlif
    : forall (n : nat) (A : Type),
      merely (A <~> Fin n) -> merely (dtb_dlib n A <~> Fin n.+1).
  Proof.
    simpl; intros n A; unfold dtb_dlib, add.
    srapply @Trunc_rec; intro e.
    exact (tr (exp e)).
  Qed.

  Definition dtb_dltwb
    : forall (n : nat) (A : Type),
      dtb_dlib n.+1 (dtb_dlib n A) = dtb_dlib n.+1 (dtb_dlib n A).
  Proof.
    intros; unfold dtb_dlib, add.
    exact (gamma A).
  Defined.

  Definition dtb_dldob
    : forall (n : nat) (A : Type), dtb_dltwb n A @ dtb_dltwb n A = idpath.
  Proof.
    intros; unfold dtb_dltwb.
    apply gamma_double.
  Defined.

  Definition dtb_dlbrb
    : forall (n : nat) (A : Type),
      (dtb_dltwb n.+1 (dtb_dlib n A) @ ap (dtb_dlib n.+2) (dtb_dltwb n A))
      @ dtb_dltwb n.+1 (dtb_dlib n A)
      = (ap (dtb_dlib n.+2) (dtb_dltwb n A) @ dtb_dltwb n.+1 (dtb_dlib n A))
        @ ap (dtb_dlib n.+2) (dtb_dltwb n A).
  Proof.
```

```
    intros; unfold dtb_dlib, dtb_dltwb.
    apply gamma_triple.
  Defined.

Definition deloop_to_BS
  : forall n : nat, deloop n -> BS n
  := deloop_rec_subuniverse
      (fun n A => merely (A <~> Fin n)) BS_trunc _
      dtb_dlpt0b dtb_dlpt0f dtb_dlib dtb_dlif dtb_dltwb dtb_dldob dtb_dlbrb.

Definition deloop_to_BSb
  : forall n : nat, deloop n -> Type
  := fun n a => (deloop_to_BS n a).1.
```

where add is add from Definition 2.42 and exp is incr from Definition 2.100, while gamma is described in Remark 5.16 and `gamma_double`, `gamma_triple` are as in the construction in Definition 5.39 itself.

Canonical finite types $[n]$ are already implemented (as `Fin n`) in the HoTT library. In Remark 5.45, we rebase canonical finite types to $[\![n]\!] :\equiv \mathfrak{f}^{\mathsf{b}}_n(\mathsf{pt}_n)$:

```
Definition FFin (n : nat)
  : Type
  := deloop_to_BSb n (dlpt n).

Lemma FFin_Fin
  : forall n : nat, FFin n = Fin n.
Proof.
  induction n.
  + constructor.
  + exact (ap add IHn).
Defined.
```

Finally, the strategy to prove that $\mathfrak{f}_n$ is an equivalence for every $n : \mathbb{N}$ is implemented as follows. Lemma 5.54 is formalized below:

```
Theorem condition_equiv_deloop_to_BS (n : nat)
  : (forall (a : deloop n) (p : deloop_to_BSb n a = FFin n),
      {q : a = dlpt n & ap (deloop_to_BSb n) q = p})
    -> IsEquiv (deloop_to_BS n).
Proof.
  intro h.
  srapply @isequiv_fcontr.
  refine (forall_BS_hprop n _ _ (deloop_to_BS n (dlpt n)) _);
  hnf.
  srapply @contr_sigma_prop;
  change (Contr {a : deloop n & deloop_to_BSb n a = FFin n}).
  srapply @Build_Contr.
  { exact (dlpt n; idpath). }
  intros [a p]. symmetry. srapply @path_sigma; simpl.
  + exact (h a p).1.
  + refine (@transport_paths_Fl (deloop n) _ (deloop_to_BSb n) _ _ (FFin n) (h a p).1 p
      @ _);
    apply moveR_Vp; refine (_ @ (concat_p1 _)^).
```

```
    exact (h a p).2^.
Defined.
```

where `forall_BS_hprop` is the term proving that we can use connectedness of $\mathcal{B}\mathsf{S}_n$ to produce a dependent function into a family of (-1)-types, and `contr_sigma_prop` generalizes (5.57).

The combinatorial machinery described in the proofs of Lemmata 5.63, 5.64 and 5.67 involves a number of proofs by induction, which we omit here. We report the implementation of the function $\mathfrak{m} : \Pi\,(n : \mathbb{N})\,.\,[\![n]\!] \to ([\![n]\!] = [\![n]\!])$ from Definition 5.70:

```
Definition move {n : nat}
  : FFin n -> FFin n = FFin n.
Proof.
  intro i; induction n.
  + exact idpath.
  + clear IHn. induction n.
    - exact idpath.
    - clear IHn. change (FFin n.+1 + Unit)%type in i. induction i as [j|[]].
      * induction n.
        ++ exact (gamma Empty).
        ++ change (FFin n.+1 + Unit)%type in j. induction j as [k|[]].
          -- exact (gamma (FFin n.+1) @ ap add (IHn k) @ gamma (FFin n.+1)).
          -- exact (gamma (FFin n.+1)).
      * exact idpath.
Defined.
```

References

[AAD07] A. Abel, K. Aehlig, and P. Dybjer, "Normalization by Evaluation for Martin-Löf Type Theory with One Universe", Electronic Notes in Theoretical Computer Science 173 (2007), Proceedings of the 23rd Conference on the Mathematical Foundations of Programming Semantics (MFPS XXIII), pp. 17–39, DOI: doi.org/10.1016/j.entcs.2007.02.025.

[ABD96] S. Agerholm, I. Beylin, and P. Dybjer, "A comparison of HOL and ALF formalizations of a categorical coherence theorem", *Theorem Proving in Higher Order Logics*, Springer Berlin Heidelberg, 1996, pp. 17–32.

[Acc17] M. Acclavio, "A Constructive Proof of Coherence for Symmetric Monoidal Categories Using Rewriting", arXiv e-print (math.CT), 2017, arXiv:1606.01722.

[Ada78] J. F. Adams, *Infinite loop spaces*, vol. no. 90, Annals of mathematics studies, Princeton University Press, 1978.

[Agda] G. Brunerie, K.-B. Hou (Favonia), E. Cavallo, T. Baumann, E. Finster, J. Cockx, C. Sattler, C. Jeris, M. Shulman, et al., *Homotopy Type Theory in Agda*, URL: https://github.com/HoTT/HoTT-Agda.

[AH56] J. F. Adams and P. J. Hilton, "On the chain algebra of a loop space", Commentarii mathematici Helvetici 30.1 (1956), pp. 305–330.

[Alt+18] T. Altenkirch, P. Capriotti, G. Dijkstra, N. Kraus, and F. Nordvall Forsberg, "Quotient Inductive-Inductive Types", *Foundations of Software Science and Computation Structures*, Springer International Publishing, 2018, pp. 293–310.

[Ann+19] D. Annenkov, P. Capriotti, N. Kraus, and C. Sattler, "Two-Level Type Theory and Applications", arXiv e-print (cs.LO), 2019, arXiv:1705.03307.

[AW09] S. Awodey and M. A. Warren, "Homotopy theoretic models of identity types", Math. Proc. Cambridge Philos. Soc. 146.1 (2009), pp. 45–55, DOI: 10.1017/S0305004108001783.

[Bau+17] A. Bauer, J. Gross, P. L. Lumsdaine, M. Shulman, M. Sozeau, and B. Spitters, "The HoTT Library: A Formalization of Homotopy Type Theory in Coq", *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, Association for Computing Machinery, 2017, pp. 164–172, DOI: 10.1145/3018610.3018615, URL: https://doi.org/10.1145/3018610.3018615.

[BCH14] M. Bezem, T. Coquand, and S. Huber, "A Model of Type Theory in Cubical Sets", *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, vol. 26, Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2014, pp. 107–128, DOI: 10.4230/LIPIcs.TYPES.2013.107.

[BD96] I. Beylin and P. Dybjer, "Extracting a proof of coherence for monoidal categories from a proof of normalization for monoids", *Types for Proofs and Programs*, Springer Berlin Heidelberg, 1996, pp. 47–61.

[Bey97] I. Beylin, "An ALF Proof of Mac Lane's Coherence Theorem", Licensiate Thesis (Revision: 5.26), Department of Computing Science, Chalmers / Göteborg University, 1997.

[Bro07] L. E. J. Brouwer, "Over De Grondslagen Der Wiskunde", dut, PhD Thesis, Universiteit van Amsterdam, 1907.

[Bro08] L. E. J. Brouwer, "De onbetrouwbaarheid der logische principes", dut, Tijdschrift voor Wijsbegeerte 2 (1908), pp. 152–158.

[Bru16] G. Brunerie, "On the homotopy groups of spheres in homotopy type theory", arXiv e-print (math.AT), 2016, arXiv:1606.05916.

[Buc19] U. Buchholtz, "Higher Structures in Homotopy Type Theory", *Reflections on the Foundations of Mathematics: Univalent Foundations, Set Theory and General Thoughts*, Springer International Publishing, 2019, pp. 151–172, DOI: 10.1007/978-3-030-15655-8_7.

[Buc20] U. Buchholtz, "Higher Algebra in Homotopy Type Theory", Formal Methods in Mathematics / Lean Together 2020, URL: https://www2.mathematik.tu-darmstadt.de/~buchholtz/fmm-leantogether.pdf, accessed: 20.05.2021.

[BvDR18] U. Buchholtz, F. van Doorn, and E. Rijke, "Higher Groups in Homotopy Type Theory", arXiv e-print (cs.LO), 2018, arXiv:1802.04315.

[CF19] V. Choudhury and M. Fiore, "The finite-multiset construction in HoTT", International Conference on Homotopy Type Theory (HoTT 2019) at Carnegie Mellon University, URL: https://hott.github.io/HoTT-2019/conf-slides/Choudhury.pdf, accessed: 20.05.2021.

[CFC58] H. B. Curry, R. Feys, and W. Craig, *Combinatory Logic: Volume I*, Studies in logic and the foundations of mathematics, North-Holland Pub. Co, 1958.

[CM95] G. Carlsson and R. J. Milgram, "Stable Homotopy and Iterated Loop Spaces", *Handbook of Algebraic Topology*, North-Holland, 1995, pp. 505–583, DOI: 10.1016/B978-044481779-2/50014-6.

[Coh+18] C. Cohen, T. Coquand, S. Huber, and A. Mörtberg, "Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom", *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, vol. 69, Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 5:1–5:34, DOI: 10.4230/LIPIcs.TYPES.2015.5.

[Coq] *INRIA - The Coq Proof Assistant*, URL: https://coq.inria.fr/.

[Coq17] T. Coquand, "Type Theory and Formalisation of Mathematics", *Computer Science – Theory and Applications*, Springer International Publishing, 2017, pp. 1–6.

[Dan12] N. A. Danielsson, "Positive h-levels are closed under W", blog post, URL: https://homotopytypetheory.org/2012/09/21/positive-h-levels-are-closed-under-w/, accessed: 20.05.2021.

[DF02] P. Dybjer and A. Filinski, "Normalization and Partial Evaluation", *Applied Semantics*, Springer Berlin Heidelberg, 2002, pp. 137–192.

[EH62] B. Eckmann and P. J. Hilton, "Group-like structures in general categories I multiplications and comultiplications", Mathematische annalen 145.3 (1962), pp. 227–255.

[Eps66] D. B. A. Epstein, "Functors between tensored categories", Inventiones mathematicae 1.3 (1966), pp. 221–228.

[FH18] K.-B. Hou (Favonia) and R. Harper, "Covering Spaces in Homotopy Type Theory", *22nd International Conference on Types for Proofs and Programs (TYPES 2016)*, vol. 97, Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 11:1–11:16, DOI: 10.4230/LIPIcs.TYPES.2016.11.

[Fin+21] E. Finster, S. Mimram, M. Lucas, and T. Seiller, "A cartesian (2,1)-category of homotopy polynomial functors in groupoids", Unpublished manuscript, URL: http://www.lix.polytechnique.fr/Labo/Samuel.Mimram/docs/mimram_polygpd.pdf.

[Fio+08] M. Fiore, N. Gambino, M. Hyland, and G. Winskel, "The cartesian closed bicategory of generalised species of structures", Journal of the London Mathematical Society 77.1 (2008), pp. 203–220, DOI: 10.1112/jlms/jdm096.

[Fra73] A. A. Fraenkel, *Foundations of set theory*, 2nd rev. ed., vol. 67, Studies in logic and the foundations of mathematics, North-Holland, 1973.

[Fru+18] D. Frumin, H. Geuvers, L. Gondelman, and N. van der Weide, "Finite Sets in Homotopy Type Theory", *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2018, Association for Computing Machinery, 2018, pp. 201–214, DOI: 10.1145/3167085.

[FS19] M. Fiore and P. Saville, "A type theory for cartesian closed bicategories", arXiv e-print (cs.lo), 2019, arXiv:1904.06538.

[Gep19] D. Gepner, "An Introduction to Higher Categorical Algebra", arXiv e-print (math.AT), 2019, arXiv:1907.02904.

[GJ90] E. Getzler and J. D. S. Jones, "A_∞-algebras and the cyclic bar complex", Illinois J. Math. 34.2 (1990), pp. 256–283, DOI: 10.1215/ijm/1255988267.

[Gor91] M. Gordon, "Introduction to the HOL System", *1991 International Workshop on the HOL Theorem Proving System and Its Applications*, 1991, pp. 2–3, DOI: 10.1109/HOL.1991.596265.

[Gyl20] H. R. Gylterud, "Multisets in type theory", Mathematical Proceedings of the Cambridge Philosophical Society 169.1 (2020), pp. 1–18, DOI: 10.1017/S0305004119000045.

[Hoq] *The HoTT library*, URL: https://github.com/HoTT/HoTT.

[HS98] M. Hofmann and T. Streicher, "The groupoid interpretation of type theory", *Twenty-five years of constructive type theory (Venice, 1995)*, vol. 36, Oxford Logic Guides, Oxford Univ. Press, 1998, pp. 83–111.

[Jac51] N. Jacobson, "Semi-Groups and Groups", *Lectures in Abstract Algebra I: Basic Concepts*, Springer New York, 1951, pp. 15–48, DOI: 10.1007/978-1-4684-7301-8_2.

[Joy81] A. Joyal, "Une théorie combinatoire des séries formelles", fre, Advances in Mathematics 42.1 (1981), pp. 1–82, DOI: 10.1016/0001-8708(81)90052-9.

[JS86] A. Joyal and R. Street, "Braided Monoidal Categories", Macquarie Mathematics Reports No. 860081 (1986).

[JS93] A. Joyal and R. Street, "Braided Tensor Categories", Advances in Mathematics 102.1 (1993), pp. 20–78, DOI: 10.1006/aima.1993.1055.

[KA18] N. Kraus and T. Altenkirch, "Free Higher Groups in Homotopy Type Theory", Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (2018), DOI: 10.1145/3209108.3209183.

[Kel64] G. M. Kelly, "On MacLane's conditions for coherence of natural associativities, commutativities, etc.", Journal of Algebra 1.4 (1964), pp. 397–402, DOI: 10.1016/0021-8693(64)90018-3.

[KK18] A. Kaposi and A. Kovács, "A Syntax for Higher Inductive-Inductive Types", *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*, vol. 108, Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 20:1–20:18, DOI: 10.4230/LIPIcs.FSCD.2018.20.

[KL18] C. Kapulkin and P. L. Lumsdaine, "The Simplicial Model of Univalent Foundations (after Voevodsky)", arXiv e-print (math.LO), 2018, arXiv:1211.2851.

[Kra15] N. Kraus, "The General Universal Property of the Propositional Truncation", *20th International Conference on Types for Proofs and Programs (TYPES 2014)*, vol. 39, Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015, pp. 111–145, DOI: 10.4230/LIPIcs.TYPES.2014.111.

[Kra16] N. Kraus, "Constructions with Non-Recursive Higher Inductive Types", *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, Association for Computing Machinery, 2016, pp. 595–604, DOI: 10.1145/2933575.2933586.

[KT08] C. Kassel and V. Turaev, *Braids and Braid Groups*, Springer New York, 2008, DOI: 10.1007/978-0-387-68548-9.

[Laf03] Y. Lafont, "Towards an algebraic theory of Boolean circuits", Journal of Pure and Applied Algebra 184.2 (2003), pp. 257–310, DOI: 10.1016/S0022-4049(03)00069-0.

[Lap72] M. L. Laplaza, "Coherence for distributivity", *Coherence in Categories*, Springer Berlin Heidelberg, 1972, pp. 29–65.

[LB15] D. R. Licata and G. Brunerie, "A Cubical Approach to Synthetic Homotopy Theory", *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2015, pp. 92–103.

[Lei04] T. Leinster, *Higher operads, higher categories*, vol. 298, London Mathematical Society lecture note series, Cambridge University Press, 2004.

[Lei14] T. Leinster, "Rethinking Set Theory", The American Mathematical Monthly 121.5 (2014), pp. 403–415.

[Lic11] D. R. Licata, "Running Circles Around (In) Your Proof Assistant; or, Quotients that Compute", blog post, URL: https://homotopytypetheory.org/2011/04/23/running-circles-around-in-your-proof-assistant/, accessed: 20.05.2021.

[LS13] D. R. Licata and M. Shulman, "Calculating the Fundamental Group of the Circle in Homotopy Type Theory", *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '13, IEEE Computer Society, 2013, pp. 223–232, DOI: 10.1109/LICS.2013.28.

[LS19] P. L. Lumsdaine and M. Shulman, "Semantics of higher inductive types", Mathematical Proceedings of the Cambridge Philosophical Society (2019), pp. 1–50, DOI: 10.1017/s030500411900015x.

[Lur17] J. Lurie, "Higher Algebra", version dated September 18, 2017, URL: http://people.math.harvard.edu/~lurie/papers/HA.pdf, accessed: 20.05.2021.

[M-L75] P. Martin-Löf, "An Intuitionistic Theory of Types: Predicative Part", *Logic Colloquium '73*, vol. 80, Studies in Logic and the Foundations of Mathematics, Elsevier, 1975, pp. 73–118, DOI: 10.1016/S0049-237X(08)71945-1.

[Mag95] L. Magnusson, "The Implementation of ALF - a Proof Editor based on Martin-Löf's Monomorphic Type Theory with Explicit Substitution", PhD Thesis, Göteborg University and Chalmers University of Technology, 1995.

[May72] J. P. May, *The Geometry of Iterated Loop Spaces*, 1st ed. 1972., vol. 271, Lecture Notes in Mathematics, Springer Berlin Heidelberg : Imprint: Springer, 1972.

[ML63] S. Mac Lane, "Natural associativity and commutativity", Rice University Studies 49.4 (1963), pp. 28–46.

[ML76] S. Mac Lane, "Topology and logic as a source of algebra", Bull. Amer. Math. Soc. 82.1 (1976), pp. 1–40, DOI: 10.1090/S0002-9904-1976-13928-6.

[ML98] S. Mac Lane, *Categories for the working mathematician*, 2nd ed., vol. 5, Graduate texts in mathematics, Springer, 1998.

[MSS02] M. Markl, S. Shnider, and J. Stasheff, *Operads in Algebra, Topology, and Physics*, vol. 96, Mathematical surveys and monographs, American Mathematical Society, 2002.

[NG14] R. Nederpelt and H. Geuvers, *Type Theory and Formal Proof: An Introduction*, Cambridge University Press, 2014, DOI: 10.1017/CBO9781139567725.

[Pic20] S. Piceghello, "Coherence for Monoidal Groupoids in HoTT", *25th International Conference on Types for Proofs and Programs (TYPES 2019)*, vol. 175, Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 8:1–8:20, DOI: 10.4230/LIPIcs.TYPES.2019.8.

[RS17] E. Riehl and M. Shulman, "A type theory for synthetic ∞-categories", Higher Structures 1(1) (2017), pp. 147–224.

[Sav20] P. Saville, "Cartesian closed bicategories: type theory and coherence", arXiv e-print (math.CT), 2020, arXiv:2007.00624.

[Shu15] M. Shulman, "The Univalent Perspective on Classifying Spaces", blog post, URL: https://golem.ph.utexas.edu/category/2015/01/the_univalent_perspective_on_c.html, accessed: 20.05.2021.

[Shu17a] M. Shulman, "Brouwer's fixed-point theorem in real-cohesive homotopy type theory", arXiv e-print (math.CT), 2017, arXiv:1509.07584.

[Shu17b] M. Shulman, "Homotopy Type Theory: A Synthetic Approach to Higher Equalities", *Categories for the Working Philosopher*, Oxford University Press, 2017, pp. 36–57.

[Soj15] K. Sojakova, "Higher Inductive Types as Homotopy-Initial Algebras", SIGPLAN Not. 50.1 (2015), pp. 31–42, DOI: 10.1145/2775051.2676983.

[Soj16] K. Sojakova, "The Equivalence of the Torus and the Product of Two Circles in Homotopy Type Theory", ACM Trans. Comput. Logic 17.4 (2016), DOI: 10.1145/2992783.

[Sta63a] J. D. Stasheff, "Homotopy associativity of H-spaces. I", Transactions of the American Mathematical Society 108.2 (1963), pp. 275–292.

[Sta63b] J. D. Stasheff, "Homotopy Associativity of H-Spaces. II", Transactions of the American Mathematical Society 108.2 (1963), pp. 293–312.

[Str93] T. Streicher, "Investigations Into Intensional Type Theory", Habilitation Thesis, Ludwig-Maximilians-Universität München, 1993.

[Tro11] A. S. Troelstra, "History of constructivism in the 20th century", *Set Theory, Arithmetic, and Foundations of Mathematics*, 2011, pp. 150–179.

[Uni13] The Univalent Foundations Program, *Homotopy Type Theory: Univalent Foundations of Mathematics*, URL: https://homotopytypetheory.org/book, 2013.

[UniMath] *The UniMath library*, URL: https://github.com/UniMath/UniMath.

[vAS15] M. van Atten and G. Sundholm, "L.E.J. Brouwer's 'Unreliability of the logical principles'. A new translation, with an introduction", arXiv e-print (math.HO), 2015, arXiv:1511.01113v1.

[vD15] F. van Doorn, "Constructing the Propositional Truncation using Non-recursive HITs", arXiv e-print (math.LO), 2015, arXiv:1512.02274.

[vD18] F. van Doorn, "On the Formalization of Higher Inductive Types and Synthetic Homotopy Theory", arXiv e-print (math.AT), 2018, arXiv:1808.10690.

[vDvRB17] F. van Doorn, J. von Raumer, and U. Buchholtz, "Homotopy Type Theory in Lean", *Interactive Theorem Proving*, Springer International Publishing, 2017, pp. 479–495.

[Voe06] V. Voevodsky, "A very short note on homotopy λ-calculus", unpublished (2006), URL: https://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/Hlambda_short_current.pdf, accessed: 20.05.2021.

[Voe14] V. Voevodsky, "The equivalence axiom and univalent models of type theory. (Talk at CMU on February 4, 2010)", arXiv e-print (math.LO), 2014, arXiv:1402.5556.

[VvdW20] N. Veltri and N. van der Weide, "Constructing Higher Inductive Types as Groupoid Quotients", arXiv e-print (cs.LO), 2020, arXiv:2002.08150.

[Whi78] G. W. Whitehead, *Elements of homotopy theory*, vol. 61, Graduate texts in mathematics, Springer, 1978.

Graphic design: Communication Division, UiB / Print: Skipnes Kommunikasjon AS

uib.no

ISBN: 9788230867723 (print)
9788230847114 (PDF)