Submitted to MFPS 22

Free Commutative Monoids in Homotopy Type Theory

Vikraman Choudhury[1] Marcelo Fiore[2,3]

Department of Computer Science and Technology, University of Cambridge

Abstract

We develop a constructive theory of finite multisets in Homotopy Type Theory, defining them as free commutative monoids. After recalling basic structural properties of the free commutative-monoid construction, we formalise and establish the categorical universal property of two, necessarily equivalent, algebraic presentations of free commutative monoids using 1-HITs. These presentations correspond to two different equational theories invariably including commutation axioms. In this setting, we prove important structural combinatorial properties of finite multisets. These properties are established in full generality without assuming decidable equality on the carrier set. As an application, we present a constructive formalisation of the relational model of classical linear logic and its differential structure. This leads to establishing that free commutative monoids are conical refinement monoids. Thereon we obtain a characterisation of the equality type of finite multisets and a new presentation of the free commutative-monoid construction as a set-quotient of the finite list construction. These developments crucially rely on the commutation relation of creation/annihilation operators associated with the free commutative-monoid construction seen as a combinatorial Fock space.

Keywords: finite-multiset construction, free commutative-monoid construction, constructive mathematics, homotopy type theory, higher inductive types, category theory, classical linear logic, differential linear logic, conical refinement monoids, combinatorial Fock space, creation/annihilation operators

1 Introduction

Martin-Löf Type Theory (MLTT) introduces the identity type to express equality in type theory. Homotopy Type Theory and Univalent Foundations (HoTT/UF) [39] embraces this proof-relevant notion of equality, and extends it with Voevodsky's univalence principle [29], homotopy types, and Higher Inductive Types (HITs), proposing a foundational system for constructive mathematics.

Algebraic structures, that is, sets with operations satisfying equations, are ubiquitous in computer science and mathematics. In HoTT, given a type A, and terms $x, y : A$, the identity type $x =_A y$ carries interesting structure. Using this equality type for equations, it is a challenging problem to define (higher) algebraic structures. For example, the problem of defining (free higher) groups has been considered by [32,11,3]. In this paper, we consider the problem of defining and studying *free commutative monoids* (on sets) in HoTT.

[1] Email: vc378@cl.cam.ac.uk
[2] Email: Marcelo.Fiore@cl.cam.ac.uk
[3] Research partially supported by EPSRC grant EP/V002309/1.

1.1 Finite multisets

A multiset is intuitively a set whose elements have multiplicities. One possible formalisation of a finite multiset on a set A is simply a finite set S with elements drawn from A, along with a multiplicity function $S \to \mathbb{N}$. Alternatively, this is informally an *unordered* list of elements drawn from A, that is, lists up to reordering of elements.

How does one define finite multisets in constructive type theory? There are several possibilities: the basic idea is to define an equivalence relation for permutations and quotient lists by this relation. This can be done either using setoids or using various techniques for constructing well-behaved quotients.

Just like lists are free monoids, finite-multisets are free commutative monoids. However, none of the existing encodings of finite-multisets in type theory prove the categorical universal property of free commutative monoids. Further, most operations on finite multisets are defined assuming decidable equality on the underlying set.

The goal of our work is to define finite multisets in HoTT, proving that they satisfy the universal property of free commutative monoids. Further, we prove several structural properties of finite multisets, without making assumptions of decidable equality on the underlying set.

1.2 Contributions

The main goal of the paper is to develop a constructive theory of free commutative monoids and their finite-multiset representation, including a characterisation of the path space, with applications to models of differential linear logic and combinatorial Fock space.

- In Section 2, we start by briefly defining commutative monoids and their homomorphisms in HoTT. Then, we state the universal property of free commutative monoids and describe the standard monad structure of the free commutative monoid on a set, which follows from its universal property. After this, we give two different constructions of the free commutative monoid using 1-HITs – the standard universal-algebraic one, and a folklore swapped-list one. We show that they both satisfy the universal property, making them equivalent.
- In Section 3, we describe the power relative monad in HoTT. We then develop the category of relations (Rel) in the spirit of Lawvere's Generalized Logic, where sets (hSets) are regarded as groupoids enriched over propositions. We exhibit the dagger compact structure of Rel, building towards the relational model of linear logic.
- In Section 4, we formalise the relational model of classical linear logic in HoTT. We use Day's promonoidal convolution to lift free commutative monoids in Set to cofree commutative comonoids in Rel, and show its linear exponential comonad structure.
- In Section 5, we formalise the differential structure of the relational model of linear logic in HoTT. To do so, we start by establishing structural properties of subsingleton multisets, and use this to characterise equalities between singleton multisets, and concatenations and projections of multisets. Using this we establish the differential structure following Fiore [22], by constructing the creation operator.
- In Section 6, we characterise the path space of finite multisets using the commutation relation of creation/annihilation operators. This leads to a sound and complete deduction system for multiset equality. Finally, using this relation as a path constructor, we introduce a new conditional equational presentation for free commutative monoids.

We have a partial formalisation of our constructions and main results using Cubical Agda. Informal paper proofs and proof sketches are also provided in the included appendices. We refer the reader to the repository for a detailed overview of the formalised proofs: https://github.com/vikraman/generalised-species/tree/master/set.

1.3 Type theory

We choose to present our results in the type-theoretic language of the HoTT book [39]. All our claims will hold in a Cubical type theory [15,4,40] as well, and our formalisation uses Cubical Agda [40]. We briefly remark on the type-theoretic notation we use in the paper.

We write dependent function types as $\prod_{x:A} B(x)$, or simply $(x : A) \to B(x)$, and sigma types as $\sum_{x:A} B(x)$, or simply $(x : A) \times B(x)$, following Agda inspired notation. Sometimes, we leave function arguments implicit, marked with braces $\{x : A\}$. We also take a few liberties with notation, when using infix, or (un)curried forms of operations, with appropriately assigned associativity.

The identity type between $x, y : A$ is written as $x =_A y$, and we write $p \circ q$ for path composition and p^{-1} for the path inverse operations. The constant path is given by $\mathsf{refl}_x : x =_A x$ and $\mathsf{ap}_f : \prod_{x,y:A} x =_A y \to f(x) =_B f(y)$ gives the functorial action on paths.

The (univalent) universe of types is denoted $\mathcal{U}$. For a type family (fibration) $B : A \to \mathcal{U}$, the $\mathsf{transport}$ operation lifts paths in the base type A to functions between the fibers, that is, $\mathsf{transport}_B : \prod_{x,y:A} x =_A y \to B(x) \to B(y)$. For $u : B(x)$ and $v : B(y)$, we write $p_\star(u)$ for $\mathsf{transport}^p_B(u)$, and $u =^B_p v$ for the lifted path over p given by $p_\star(u) =_{B(y)} v$. In Cubical Agda, this is given by the heterogeneous path type PathP. The functorial action on sections $f : \prod_{x:A} B(x)$ of the fibration is given by $\mathsf{apd}_f : \prod_{x,y:A} (p : x =_A y) \to f(x) =^B_p f(y)$.

Equivalences between types A and B are denoted by $A \simeq B$, and homotopies between functions f and g are denoted by $f \sim g$. By $\mathsf{univalence}/\mathsf{funext}$, these are equivalent to the corresponding identity types. We use $:\equiv$ for definitions and $\equiv$ for denoting judgemental equalities.

For homotopy n-types, we use the standard definitions for contractible types (-2), propositions (-1) and sets (0), which are given by $\mathsf{isContr}(A) :\equiv \sum_{x:A} \prod_{y:A} y = x$, $\mathsf{isProp}(A) :\equiv \prod_{x,y:A} \mathsf{isContr}(x = y)$, and $\mathsf{isSet}(A) :\equiv \prod_{x,y:A} \mathsf{isProp}(x = y)$. We write hProp and hSet for the universe of propositions and sets respectively. We write Set for the (univalent) category of hSets and functions [39, Chapter 10.1]. When writing commuting diagrams, we mean that they commute up to homotopy (or equivalently, up to the identity type).

The n-truncation of a type A is written as $\|A\|_n$, with the point constructor $|-| : A \to \|A\|_n$. When working with propositions ((-1)-truncated types), we use standard logical notation, that is truth values, $\bot :\equiv \mathbf{0}$, $\top :\equiv \mathbf{1}$, binary connectives $\phi \wedge \psi :\equiv \phi \times \psi$, and $\phi \vee \psi :\equiv \|\phi + \psi\|_{-1}$, and quantifiers given by $\forall_{(x:A)}.P(x) :\equiv \prod_{x:A} P(x)$ and $\exists_{(x:A)}.P(x) :\equiv \|\sum_{x:A} P(x)\|_{-1}$ (see [39, Definition 3.7.1].

2 The free commutative monoid monad

We start by giving the definition of commutative monoids in type theory; they are monoids with an additional commutation axiom.

Definition 2.1 (Commutative monoid)**.** *A commutative monoid is a set M with:* (i) *a* multiplication *function $- \cdot - : M \times M \to M$; and* (ii) *a* unit *element $e : M$; such that* (i) *for $x : M$, we have $x \cdot e = x$ $e \cdot x = x$;* (ii) *for $x, y, z : M$, we have $x \cdot (y \cdot z) = (x \cdot y) \cdot z$; and* (iii) *for $x, y : M$, we have $x \cdot y = y \cdot x$.*

Example 2.1. *The natural numbers $\mathbb{N}$ are a commutative monoid under addition, with unit 0, and also under multiplication, with unit 1. For a set A, lists $\mathcal{L}(A)$ with the empty list nil for unit, and the append operation $+\!\!+$ for multiplication are a monoid, but not generally a commutative monoid.*

Definition 2.2 (Homomorphism)**.** *For commutative monoids $(M; \cdot_M, e_M)$ and $(N; \cdot_N, e_N)$, a function $f : M \to N$ is a homomorphism if it preserves the unit and multiplication.*

$$\mathsf{isCMonHom}(f) :\equiv f(e_M) = e_N \ \wedge \ \forall(x, y : M).f(x \cdot_M y) = f(x) \cdot_N f(y)$$

The set of homomorphisms is $\mathsf{CMon}(M, N) :\equiv \sum_{f:M\to N} \mathsf{isCMonHom}(f)$.

We are mainly interested in free commutative monoids whose definition we state next.

Definition 2.3 (Universal property of free commutative monoids)**.** *For a set A, a commutative monoid $\mathcal{M}(A)$ together with a function $\eta_A : A \to \mathcal{M}(A)$ is the free commutative monoid on A whenever, for every commutative monoid M, composition with η determines an equivalence as follows*

$$(-) \circ \eta_A : \mathsf{CMon}(\mathcal{M}(A), M) \xrightarrow{\sim} (A \to M)$$

Equivalently, every function $f : A \to M$ has a unique homomorphic extension $f^\sharp : \mathsf{CMon}(\mathcal{M}(A), M)$ along η_A; that is, the type $\sum_{h:\mathsf{CMon}(\mathcal{M}(A),M)} \pi_1(h) \circ \eta_A = f$ is contractible.

Proposition 2.4. *Let $\mathcal{L}(A)$ be the free monoid (equivalently, the list construction) on A. The free commutative monoid and the free monoid on the unit type $\mathbf{1}$ are canonically equivalent: $\mathcal{M}(\mathbf{1}) \simeq \mathcal{L}(\mathbf{1})$. Hence, $\mathcal{M}(\mathbf{1}) \simeq \mathbb{N}$.*

We have given the free commutative monoid as an endofunction $\mathcal{M}$ on the type of sets hSet. It is standard that the construction extends to a strong monad on the category of sets and functions.

Definition 2.5. *For sets A and B,*

(i) *the functorial action for a function $f : A \to B$ is $\mathcal{M}(f) :\equiv (\lambda(a:A).\, \eta_B(fa))^\sharp : \mathcal{M}(A) \to \mathcal{M}(B)$;*

(ii) *the unit and multiplications are $\eta_A : A \to \mathcal{M}(A)$ and $\mu_A :\equiv (\lambda(x:A).\, x)^\sharp : \mathcal{M}(\mathcal{M}(A)) \to \mathcal{M}(A)$;*

(iii) *the left and right tensorial strengths are $\sigma_{A,B} : \mathcal{M}(A) \times B \to \mathcal{M}(A \times B) : (as, b) \mapsto \mathcal{M}(\lambda(a:A).\, (a, b))(as)$, and $\tau_{A,B} : A \times \mathcal{M}(B) \to \mathcal{M}(A \times B) : (a, bs) \mapsto \mathcal{M}(\lambda(b:B).\, (a, b))(bs)$.*

Furthermore, the free commutative-monoid monad is commutative [31].

Proposition 2.6. *For all sets A and B, $\tau_{A,B} \circ \sigma_{A,\mathcal{M}(B)}{}^\sharp = \sigma_{A,B}{}^\sharp \circ \tau_{\mathcal{M}(A),B} : \mathcal{M}(A) \times \mathcal{M}(B) \to \mathcal{M}(A \times B)$. This map is a homomorphism in each argument and universal amongst all such.*

Example 2.2. *The free commutative monoid $\mathcal{M}(\mathbf{1})$ on $\mathbf{1}$ has the additive structure of natural numbers (Proposition 2.4). Hence, any free commutative monoid comes equipped with a length function:*

$$\ell_A :\equiv \mathcal{M}(\lambda(a:A).\, \star) : \mathcal{M}(A) \to \mathcal{M}(\mathbf{1})$$

The multiplicative structure of $\mathcal{M}(\mathbf{1})$ is given by $\eta(\star) : \mathcal{M}(\mathbf{1})$ and $\pi_2{}^\sharp \circ \sigma_{\mathbf{1},\mathcal{M}(\mathbf{1})} : \mathcal{M}(\mathbf{1}) \times \mathcal{M}(\mathbf{1}) \to \mathcal{M}(\mathbf{1})$. This is commutative by Proposition 2.6.

Proposition 2.7. *The free commutative-monoid monad is canonically a strong symmetric monoidal endofunctor from the coproduct monoidal structure to the product monoidal structure; the monoidal isomorphisms are*

$$\mathcal{M}(A) \times \mathcal{M}(B) \xrightarrow{\simeq} \mathcal{M}(A + B) \quad , \quad \mathbf{1} \xrightarrow[\mathsf{nil}]{\simeq} \mathcal{M}(\mathbf{0})$$

$$\mathcal{M}(A) \times \mathcal{M}(B) \xrightarrow[\mathcal{M}(\iota_1)\times\mathcal{M}(\iota_2)]{} \mathcal{M}(A + B) \times \mathcal{M}(A + B) \xrightarrow{+_{(A+B)}} \mathcal{M}(A + B)$$

We have described the universal property of free commutative monoids but we have not yet given a construction for them. We will describe three constructions in HoTT: (i) the naive one from universal algebra (Section 2.1); (ii) a folklore swapped-list construction from computer science (Section 2.2); and (iii) a new quotiented-list construction (Section 6.2) stemming from our study of multiset equality (Section 6.1).

The situation concerning the constructions (*i*) and (*ii*) is analogous to that of free monoids, which can either be described by generators and relations or by using the inductive list type (see [39, Chapter 6.11]); our construction (*iii*) is of independent interest. For each construction, our goal is to establish the categorical universal property; so that they automatically acquire the structure detailed in this section, albeit established in an implementation-independent manner. This is a distinguishing feature of our work compared to previous approaches to finite multisets in type theory.

2.1 Universal-algebraic construction

Using the standard construction of free algebras for equational theories from universal algebra, the free commutative monoid on a set can be defined as follows.

Definition 2.8 (FCM)**.** *Let A be a type, we define the 1-HIT $\mathsf{FCM}(A)$ with the following point and path constructors:*

$$
\begin{aligned}
\eta &: A \to \mathsf{FCM}(A) \\
e &: \mathsf{FCM}(A) \\
- \cdot - &: \mathsf{FCM}(A) \times \mathsf{FCM}(A) \to \mathsf{FCM}(A) \\
\mathsf{assoc} &: (x\ y\ z : \mathsf{FCM}(A)) \to x \cdot (y \cdot z) = (x \cdot y) \cdot z \\
\mathsf{unitl} &: (x : \mathsf{FCM}(A)) \to e \cdot x = x \\
\mathsf{unitr} &: (x : \mathsf{FCM}(A)) \to x \cdot e = x \\
\mathsf{comm} &: (x\ y : \mathsf{FCM}(A)) \to x \cdot y = y \cdot x \\
\mathsf{trunc} &: \mathsf{isSet}(\mathsf{FCM}(A))
\end{aligned}
$$

The first constructor η postulates that A maps to $\mathsf{FCM}(A)$. The next two give the operations on a monoid: the identity element and multiplication (written in an infix style). The four path constructors after that assert the axioms of a commutative monoid – associativity, unitality, and commutativity. Finally, we add a path constructor to truncate $\mathsf{FCM}(A)$ to a set, since we wish to ignore any higher paths introduced by the path constructors. For completeness, we state the induction principle for FCM (Definition A.1).

$\mathsf{FCM}(A)$ is straightforwardly a commutative monoid. As expected, using the induction principle and computation rules, one proves that it satisfies the universal property of the free commutative monoid (Definition 2.3).

Theorem 2.9. *For every set A, $\eta : A \to \mathsf{FCM}(A)$ is the free commutative monoid on A.*

2.2 Swapped-list construction

Alternatively, one can define free commutative monoids by means of a folklore *swapped-list* construction. Just like free monoids are given by finite lists (defined as an inductive type), free commutative monoids can be defined as lists, but identified up to swappings.

Definition 2.10 (sList)**.** *The swapped-list over a type A is given by the type $\mathsf{sList}(A)$, which is a (recursive) 1-HIT with the following point and path constructors:*

$$
\begin{aligned}
\mathsf{nil} &: \mathsf{sList}(A) \\
- :: - &: A \times \mathsf{sList}(A) \to \mathsf{sList}(A) \\
\mathsf{swap} &: (x\ y : A)(xs : \mathsf{sList}(A)) \to x :: y :: xs = y :: x :: xs \\
\mathsf{trunc} &: \mathsf{isSet}(\mathsf{sList}(A))
\end{aligned}
$$

The first two constructors are the standard ones for lists (free monoids), that is, nil and cons or (infix) $::$, but we have an additional swap path constructor asserting that the first two elements of any list can be swapped around. Note that, unlike FCM, this is a recursively defined HIT, that is, the points and paths are recursively generated. In particular, by applying cons on paths by congruence and path composition, these adjacent swaps can be performed anywhere inside the list. Finally, we truncate $\mathsf{sList}(A)$ to a set, since we wish to ignore the higher paths introduced by swap.

Classically, finite-multisets are usually defined as lists quotiented by permutations of their elements. In our setting, we generate this equivalence relation recursively, that is, we are generating the path space of finite multisets simply using the swap path constructor. To establish this fact formally, we prove that $\mathsf{sList}(A)$ is indeed the free commutative monoid on A by establishing its universal property.

The main means of reasoning about swapped-lists is its induction principle (Definition A.2). It is akin to the induction principle for lists, but one needs to additionally enforce invariance under swapping. Also, since we truncate to sets, one is only allowed to eliminate to sets. This is in general restrictive. However, for the purposes of this paper, we are only interested in establishing the universal property of free commutative monoids and only ever need to eliminate to sets (or propositions).

When eliminating to propositions, the induction principle can be simplified further. One only needs to provide an image for the nil and $::$ constructors, similar to the case for lists.

Lemma 2.11 (Propositional induction principle for sList)**.**
Let $P : \mathsf{sList}(A) \to \mathcal{U}$ *be a type family with the following data.*

$$\begin{aligned} \mathsf{nil}^* &: P(\mathsf{nil}) \\ - ::^* - &: (x : A)\{xs : \mathsf{sList}(A)\} \to P(xs) \to P(x :: xs) \\ \mathsf{trunc}^* &: (xs : \mathsf{sList}(A)) \to \mathsf{isProp}(P(xs)) \end{aligned}$$

Then, there is a unique function $f : (xs : \mathsf{sList}(A)) \to P(xs)$ *satisfying appropriate computation rules.*

We now exhibit the swapped-list construction as the free commutative monoid construction.

Definition 2.12 (η)**.** *The generators map is given by* $\eta : A \to \mathsf{sList}(A) : a \longmapsto [a] :\equiv (a :: \mathsf{nil})$.

We show that $\mathsf{sList}(A)$ is a commutative monoid with the empty multiset nil as unit and the binary multiplication given by the concatenation (or append) operation $+\!\!+$ defined below.

Definition 2.13 ($+\!\!+$)**.** *The concatenation operation* $- +\!\!+ - : \mathsf{sList}(A) \to \mathsf{sList}(A) \to \mathsf{sList}(A)$ *is defined by recursion on the first argument. Using the recursion principle on the point and path constructors, we have*

$$\begin{aligned} \mathsf{nil} +\!\!+ ys &:\equiv ys \\ (x :: xs) +\!\!+ ys &:\equiv x :: (xs +\!\!+ ys) \\ \mathsf{ap}_{- +\!\!+ ys}(\mathsf{swap}(x, y, xs)) &:\equiv \mathsf{swap}(x, y, xs +\!\!+ ys) \end{aligned}$$

Finally, $\mathsf{sList}(A)$ *is a set and so is* $\mathsf{sList}(A) \to \mathsf{sList}(A)$*, hence it respects* trunc.

To show that $\mathsf{sList}(A)$ is a commutative monoid, one needs to prove a few identities about elements of $\mathsf{sList}(A)$. Since $\mathsf{sList}(A)$ is a set, its equality type is a proposition, so one can use the propositional induction principle from Lemma 2.11. Hence, the proofs of the monoid laws are simply the ones for lists.

Lemma 2.14. *The concatenation operation* $+\!\!+$ *is associative and* nil *is a left and right unit. For all* $xs, ys, zs : \mathsf{sList}(A)$*, we have* $xs +\!\!+ (ys +\!\!+ zs) = (xs +\!\!+ ys) +\!\!+ zs$*,* $\mathsf{nil} +\!\!+ xs = xs$*, and* $xs +\!\!+ \mathsf{nil} = xs$.

We further need to establish that the concatenation operation is commutative. Similar to proving commutativity of addition for natural numbers, this is shown in steps as follows. For details, we refer the reader to the supplementary material in Appendix A.

Lemma 2.15. *For any* $x : A$*, and* $xs, ys : \mathsf{sList}(A)$*, we have* $x :: xs = xs +\!\!+ [x]$*, and* $xs +\!\!+ ys = ys +\!\!+ xs$.

Proposition 2.16. *For a type* A*,* $\mathsf{sList}(A)$ *is a commutative monoid.*

Proposition 2.17. *Given a set* A*, a commutative monoid* M*, and a map* $f : A \to M$ *of sets,* f *extends to a homomorphism* $f^\sharp : \mathsf{CMon}(\mathsf{sList}(A), M)$.

Theorem 2.18. *For every set* A*,* $\eta : A \to \mathsf{sList}(A)$ *is the free commutative monoid on* A.

As both FCM and sList satisfy the same categorical universal property, it follows that they are equivalent.

Corollary 2.19. *For every set* A*, we have an equivalence* $\mathsf{sList}(A) \simeq \mathsf{FCM}(A)$.

Convention. *We will henceforth use* $\mathcal{M}(A)$ *to indicate the free commutative monoid on a set* A *that will be also referred to as the* finite-multiset *construction.*

Our results for the free commutative monoid construction $\mathcal{M}$ will therefore apply to either FCM or sList using univalence and transport.

3 Generalized logic

This section, which stands in its own right, is an interlude to consider the category of relations in HoTT. This is pivotal in our subsequent study and applications of the free commutative-monoid construction to be carried out in the rest of the paper.

We consider HoTT from the viewpoint of Lawvere's Generalized Logic [34]. The latter takes place in enriched category theory and the analogy we pursue here is that of regarding hSets as groupoids *intrinsically* (weakly) enriched over propositions (or, in informal technical jargon, hProp-groupoids with the groupoid

structure provided by the identity type, with hProp considered with its complete Heyting algebra structure). Central to us, is that this turns out to provide the appropriate framework for developing the calculus of relations in HoTT.

3.1 Power objects

We begin by introducing power objects. Recalling that we work in predicative type theory, we use subscripts to indicate universe levels.

Definition 3.1 (Power construction)**.** *We define* $\mathfrak{P} : \mathsf{hSet}_i \to \mathsf{hSet}_{i+1} : A \longmapsto (A \to \mathsf{hProp}_i)$.

Note that the above is not a *powerset* construction – it is not even an endofunction on hSet_i. If so inclined, we could turn it into the powerset monad using Voevodsky's *propositional resizing axiom* [41] to lower the universe level. Instead, we choose here to work with the power construction as a *relative monad* [2,21] and follow the framework of Kleisli bicategories [26,21] to consider relations in HoTT. Henceforth, we drop the universe levels.

From the viewpoint of generalized logic, the power relative monad structure is given by the Yoneda embedding and left Kan extension along it. Internally, in HoTT, these operations are easily constructed using the identity type and (suitably truncated) dependent product types, respectively.

Definition 3.2 (Power relative monad)**.** *We define the relative monad operations for* $\mathfrak{P}$ *as follows.*

(i) *The unit is* $\yen_A : A \to \mathfrak{P}(A) : a \longmapsto \lambda(x:A).\, a =_A x$.

(ii) *The extension operation, for* $f : A \to \mathfrak{P}(B)$*, is* $f^* : \mathfrak{P}(A) \to \mathfrak{P}(B) : (\alpha, b) \longmapsto \exists_{(a:A)}. f(a,b) \wedge \alpha(a)$.

We establish the following properties of the Kleisli structure which are used later in Proposition 3.5.

Proposition 3.3. *The following identities hold.*

(i) $\yen_A{}^* = \mathsf{id}_{\mathfrak{P}(A)}$.

(ii) *For* $f : A \to \mathfrak{P}(B)$*, we have* $f = f^* \circ \yen_A$.

(iii) *For* $f : A \to \mathfrak{P}(B)$ *and* $g : B \to \mathfrak{P}(C)$*, we have that* $(g^* \circ f)^* = g^* \circ f^*$.

3.2 The category of relations

We define the category of relations (also see [39, Example 9.1.19]) as the Kleisli category of $\mathfrak{P}$.

Definition 3.4 (Rel)**.** Rel *has objects* hSet*s, and homs* $A \nrightarrow B :\equiv A \to \mathfrak{P}(B)$.

Proposition 3.5. Rel *is a (univalent) category.*

The category Rel has a symmetric monoidal structure given by the cartesian product of sets $A \otimes B :\equiv A \times B$ with unit $\mathbf{1}$. It is also closed, with $A \multimap B :\equiv A \times B$. The tensor-hom adjunction is given by:

$$C \otimes A \nrightarrow B \equiv (C \times A) \to B \to \mathsf{hProp} \simeq C \to A \to B \to \mathsf{hProp} \simeq C \to (A \times B) \to \mathsf{hProp} \equiv C \nrightarrow (A \multimap B) \ .$$

In fact, Rel is compact closed [18,30], as the canonical map $\kappa : C \otimes (A \multimap B) \longrightarrow A \multimap (C \otimes B)$ is an equivalence. Furthermore, as $A^\perp \equiv A \multimap \mathbf{1} \simeq A$, Rel is self dual, with involution given by

$$(-)^\dagger : A \nrightarrow B \equiv A \to (B \to \mathsf{hProp}) \simeq B \to (A \to \mathsf{hProp}) \equiv B \nrightarrow A \ .$$

To summarise:

Proposition 3.6. Rel *is a dagger compact category.*

Definition 3.7. *We define the inclusion* $(-)_* : \mathsf{Set} \to \mathsf{Rel}$ *as mapping functions* $f : A \to B$ *to relations* $\yen_B \circ f : A \nrightarrow B$.

Note that $(-)_*$ preserves coproducts, which in Rel become biproducts. Therefore, every set is endowed with a biproduct commutative bialgebra structure; namely, compatible coproduct commutative monoid

and product commutative comonoid structures, crucially satisfying the following identity:

$$\begin{array}{ccccc} A+A & & \xrightarrow{\nabla} & & A \xrightarrow{\Delta} A+A \\ {\scriptstyle \Delta+\Delta}\downarrow & & & & \uparrow{\scriptstyle \nabla+\nabla} \\ A+A+A+A & & \xrightarrow[\mathsf{id}_{\mathcal{M}(A)}+c+\mathsf{id}_{\mathcal{M}(A)}]{} & & A+A+A+A \end{array} \tag{1}$$

where c is the symmetry isomorphism.

4 Relational model of linear logic

We give a formalisation of the relational model of linear logic (see, in particular, Hyland [27, Section 3]) developing it in HoTT from the perspective of generalized logic considered in the previous section. We establish that the symmetric monoidal closed category with finite biproducts Rel admits cofree commutative comonoids, and is therefore a model of linear logic with linear exponential comonad structure [7,28] as first considered by [33].

The key to the development is the lifting of (commutative) monoid structure to power objects. This is achieved by means of Day's *promonoidal convolution* [17].

Theorem 4.1 (Promonoidal convolution)**.** *If $m : M \otimes M \nrightarrow M \nleftarrow \mathbf{1} : e$ is a (commutative) monoid in Rel, then $\mathfrak{P}(M)$ is a (commutative) monoid in Set, with multiplication $\hat{m} : \mathfrak{P}(M) \times \mathfrak{P}(M) \to \mathfrak{P}(M)$ and unit $\hat{e} : \mathfrak{P}(M)$ given by*

$$\begin{aligned} \hat{m}(p,q)(x) &:\equiv \exists_{(y:M)}.\exists_{(z:M)}.p(y) \wedge q(z) \wedge m(y,z)(x) \\ \hat{e}(x) &:\equiv e(\star)(x) \ . \end{aligned}$$

Example 4.1. *Since the inclusion $(-)_* : \mathsf{Set} \to \mathsf{Rel}$ maps products to tensors, every (commutative) monoid $(M, \cdot, e)$ in Set yields a (commutative) monoid in Rel that, by promonoidal convolution (Theorem 4.1), produces a (commutative) monoid $(\mathfrak{P}(M), \hat{\cdot}, \hat{e})$ in Set. In concrete terms,*

$$\begin{aligned} (p \mathbin{\hat{\cdot}} q)(x) &\equiv \exists_{(y:M)}.\exists_{(z:M)}.p(y) \wedge q(z) \wedge (y \cdot z = x) \\ \hat{e}\,(x) &\equiv (e = x) \ . \end{aligned}$$

The universal property of $\mathcal{M}$ lifts from Set to Rel.

Theorem 4.2. *For every set A, the set $\mathcal{M}(A)$ with multiplication $m :\equiv (+\!\!+_A)_* : \mathcal{M}(A) \otimes \mathcal{M}(A) \nrightarrow \mathcal{M}(A)$ and unit $e :\equiv (\lambda(x:\mathbf{1}).\,\mathsf{nil})_* : \mathbf{1} \nrightarrow \mathcal{M}(A)$ equipped with $(\eta_A)_* : A \nrightarrow \mathcal{M}(A)$ is the free commutative monoid on A in Rel.*

We turn our attention to the dual algebraic structure of commutative comonoid.

Definition 4.3 (Commutative comonoid)**.** *A commutative comonoid in Rel is a set K with: a* comultiplication *$w : K \nrightarrow K \otimes K$ and a* counit *$k : K \nrightarrow \mathbf{1}$ satisfying the following commutative diagrams (up to homotopy):*

$$\begin{array}{ccccc} \mathbf{1} \otimes K & \xrightarrow{k \otimes K} & K & \xleftarrow{K \otimes k} & K \otimes \mathbf{1} \\ & \simeq\searrow & \downarrow{\scriptstyle w} & \swarrow\simeq & \\ & & K \otimes K & & \end{array} \qquad \begin{array}{ccc} K & \xrightarrow{w} & K \otimes K \\ {\scriptstyle w}\downarrow & & \downarrow{\scriptstyle K \otimes w} \\ K \otimes K & \xrightarrow[w \otimes K]{} & K \otimes K \otimes K \end{array} \qquad \begin{array}{ccc} & & K \otimes K \\ & {\scriptstyle w}\nearrow & \\ K & & \simeq\downarrow{\scriptstyle c} \\ & {\scriptstyle w}\searrow & \\ & & K \otimes K \end{array}$$

where c is the symmetry isomorphism.

Example 4.2. *Each set has a unique product commutative comonoid structure in Set. This is preserved by the inclusion $(-)_* : \mathsf{Set} \to \mathsf{Rel}$ giving a commutative comonoid structure on the set in Rel. This is the unique commutative comonoid structure on $\mathbf{0}$ and $\mathbf{1}$. In Rel, by self-duality, every (commutative) monoid induces a (commutative) comonoid, and vice versa.*

Definition 4.4 (Homomorphism)**.** *For commutative comonoids (K, w, k) and (K', w', k'), a relation $r : K \nrightarrow K'$ is a homomorphism whenever $w' \circ r = (r \otimes r) \circ w$ and $k' \circ r = k$.*

Corollary 4.5. *For every set A, $\epsilon_A :\equiv ((\eta_A)_*)^\dagger : \mathcal{M}(A) \nrightarrow A$ is the cofree commutative comonoid on A in* Rel*; that is, for a commutative comonoid K, every relation $r : K \nrightarrow A$ has a unique homomorphic coextension $r_\sharp : K \nrightarrow \mathcal{M}(A)$ over ϵ_A.*

Theorem 4.6. Rel *is a model of linear logic. The linear exponential comonad structure is as follows:*

comonad structure

$$\delta_A :\equiv ((\mu_A)_*)^\dagger : \mathcal{M}(A) \nrightarrow \mathcal{M}(\mathcal{M}(A))$$
$$\epsilon_A :\equiv ((\eta_A)_*)^\dagger : \mathcal{M}(A) \nrightarrow A$$

monoidal structure

$$\varphi_{A,B} :\equiv (\epsilon_A \otimes \epsilon_B)_\sharp : \mathcal{M}(A) \otimes \mathcal{M}(B) \nrightarrow \mathcal{M}(A \otimes B)$$
$$\phi :\equiv (\mathrm{id}_{\mathbf{1}})_\sharp : \mathbf{1} \nrightarrow \mathcal{M}(\mathbf{1})$$

commutative comonoid structure

$$w_A :\equiv ((+\!\!+_A)_*)^\dagger : \mathcal{M}(A) \nrightarrow \mathcal{M}(A) \otimes \mathcal{M}(A)$$
$$k_A :\equiv ((\lambda(x : \mathbf{1}).\, \mathsf{nil})_*)^\dagger : \mathcal{M}(A) \nrightarrow \mathbf{1}$$

The finite-multiset construction is canonically a strong symmetric monoidal endofunctor on Rel from the coproduct monoidal structure to the product monoidal structure (recall Proposition 2.7); the monoidal isomorphisms are the Seely (or storage) isomorphisms [38]:

$$\mathcal{M}(A) \otimes \mathcal{M}(B) \xrightarrow{\simeq} \mathcal{M}(A + B) \ , \quad \mathbf{1} \xrightarrow{\simeq} \mathcal{M}(\mathbf{0}) \ . \tag{2}$$

5 Relational model of differential linear logic

We now give a formalisation of the differential structure of the relational model of linear logic [19]. We need to first establish combinatorial properties of subsingleton multisets. As we will see in Section 6, these structural properties are also needed to characterise the equality type of finite multisets.

5.1 Subsingleton multisets

We start by describing empty and singleton multisets. These can be characterised by their lengths (recall Example 2.2), that is, a multiset as is empty if it has length 0, $\ell(as) = 0$, and it is a singleton if it has length 1, $\ell(as) = 1$. Alternatively, one can say that a multiset is empty if it is equal to nil, and it is a singleton if it is equal to a one-element multiset. We will show that these are equivalent notions.

Definition 5.1. *For $as : \mathcal{M}(A)$, we define* $\mathsf{isEmpty}(as) :\equiv (as = \mathsf{nil})$ *and* $\mathsf{isSing}(as) :\equiv \sum_{a:A} (as = [a])$.

Since the equality type of $\mathcal{M}(A)$ is a proposition, it follows that $\mathsf{isEmpty}(as)$ is a proposition. It is easy to see that this is equivalent to being of length 0.

Proposition 5.2. *For $as : \mathcal{M}(A)$,* $\mathsf{isEmpty}(as) \Leftrightarrow (\ell(as) = 0)$.

It follows that free commutative monoids are *conical*, that is, nil is the only invertible element:

Corollary 5.3 (Conical-monoid relation)**.** *For $as, bs : \mathcal{M}(A)$, $as +\!\!+ bs = \mathsf{nil} \Leftrightarrow as = bs = \mathsf{nil}$.*

Note that the $a : A$ in $\mathsf{isSing}(as)$ is not truncated and hence that this type is not obviously a proposition. We can nevertheless show that there is a *unique choice* for such an a, making it a proposition. Starting from length-one multisets, one can construct a unique choice function that extracts the only element by induction on the structure of $\mathsf{sList}(A)$. This allows us to establish the following results. For details, we refer the reader to the supplementary material in Appendix D.

Proposition 5.4. *Let A be a set.*

(i) *There is a unique choice function* $\mathsf{uc} : (as : \mathcal{M}(A)) \to (\ell(as) = 1) \to \mathsf{isSing}(as)$.

(ii) *The universal map* $\eta_A : A \to \mathcal{M}(A)$ *is an embedding; that is,* $\mathsf{ap}_{\eta_A} : x = y \to [x] = [y]$ *is an equivalence.*

(iii) *For* $as : \mathcal{M}(A)$, $\mathsf{isSing}(as)$ *is a proposition.*

(iv) *For* $as : \mathcal{M}(A)$, *we have that* $\mathsf{uc}(as)$ *is an equivalence.*

Using the above, we show that the type of singleton (or equivalently, length-one) multisets is equivalent to the underlying set.

Proposition 5.5. *For a set* A,

$$A \;\simeq\; \textstyle\sum_{as:\mathcal{M}(A)} (\ell(as) = 1) \;\simeq\; \sum_{as:\mathcal{M}(A)} \mathsf{isSing}(as) \;\; .$$

Finally, we establish structural properties of singleton multisets arising from concatenations and projections. These combinatorial properties will play a central role in Section 5.2.

Proposition 5.6. *Let* A *be a set.*

(i) *For* $s : \mathcal{M}(\mathcal{M}(A))$ *and* $a : A$, $\mu(s) = [a] \Leftrightarrow \exists_{(t:\mathcal{M}(\mathcal{M}(A)))}.\mu(t) = \mathsf{nil} \wedge [a] :: t = s$.

(ii) *For a set* B *and* $t : \mathcal{M}(A \times B)$, $a : A$, $\mathcal{M}(\pi_1)(t) = [a] \;\Leftrightarrow\; \exists_{(b:B)}.t = [(a, b)]$.

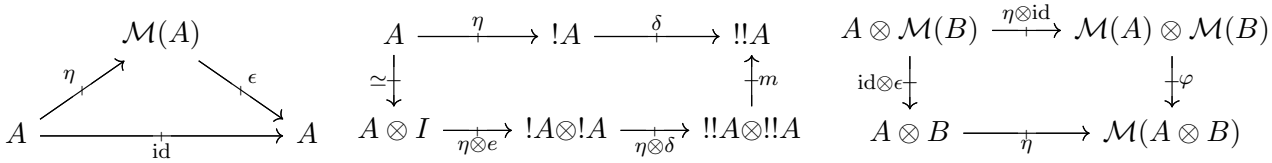
(iii) *For* $as, bs : \mathcal{M}(A)$ *and* $a : A$, $as +\!\!+ bs = [a] \Leftrightarrow (as = [a] \wedge bs = \mathsf{nil}) \vee (as = \mathsf{nil} \wedge bs = [a])$.

5.2 Differential structure

Differential structure in categorical models of linear logic can be described in a variety of forms, see for instance [8,22,19]. Here, we follow the presentation of Fiore [22] in the context of models with biadditive structure (that is, biproducts) as in Section 4. In Rel, this differential structure consists of a *creation map* [22, Definition 3.4]:

$$\eta : A \nrightarrow \mathcal{M}(A)$$

subject to three laws as follows:



that as explained in [8], induce a *differential category* [9] structure. In elementary terms, the above diagrams amount to the properties below, that follow from the results of Section 5.1.

Proposition 5.7.

(i) *For* $a, a' : A$,

$$a = a' \iff [a] = [a'] \;\; .$$

(ii) *For* $a : A$ *and* $s : \mathcal{M}(\mathcal{M}(A))$,

$$[a] = \mu(s) \iff \exists_{(t:\mathcal{M}(\mathcal{M}(A)))}.\; \mu(t) = \mathsf{nil} \wedge [a] :: t = s \;\; .$$

(iii) *For* $a : A$, $bs : \mathcal{M}(B)$ *and* $ps : \mathcal{M}(A \times B)$,

$$[a] = \mathcal{M}(\pi_1)(ps) \wedge bs = \mathcal{M}(\pi_2)(ps) \iff \exists_{(b:B)}.\; bs = [b] \wedge \big[(a, b)\big] = ps \;\; .$$

To summarise:

Theorem 5.8. Rel *is a model of differential linear logic.*

6 Path space of finite multisets

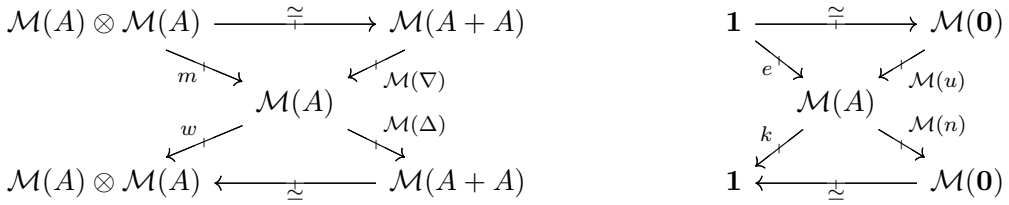
An application of our development is an understanding of multiset equality. We show that free commutative monoids are refinement monoids (Proposition 6.2) and, from there, characterise the path space of free

commutative monoids for non-empty multisets by means of a *commutation relation* (Theorem 6.3). Then, we further use this to characterise the path space of finite multisets (Theorem 6.6) and to give a deduction system for multiset equality (Section 6.2).

6.1 Commutation relation

Two non-empty multisets of the form $x :: xs$ are equal if they are either equal point-wise, by the congruence of cons, or are equal up to a permutation. Instead of explicitly working with permutations, we will characterise this equality using a commutation relation that stems from the theory of creation/annihilation operators associated with the finite-multiset construction seen as a combinatorial Fock space. We refer the reader to Fiore [22,20] for this theory.

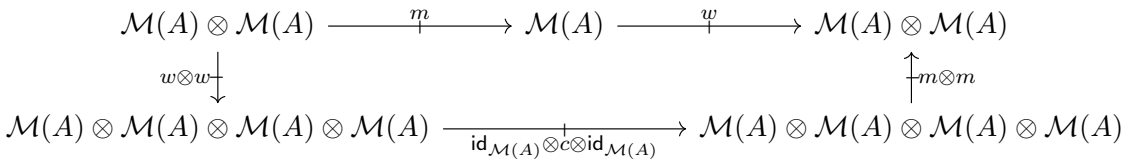
The following diagrams, involving the Seely isomorphisms (2),



commute (up to homotopy) and one can use the symmetric monoidal coherence laws to transport the biproduct commutative bialgebra structure through $\mathcal{M}$ as follows, see [22,20].

Proposition 6.1. *The biproduct commutative bialgebra structure (∇, Δ, u, n) on a set A in* Rel *transfers to one on $\mathcal{M}(A)$ as (m, w, e, k) for (m, e) and (w, k) respectively defined as in Theorems 4.2 and 4.6.*

In particular, the bialgebra identity (1) transports to the one below:



Spelling this out in elementary terms, we have that free commutative monoids satisfy the *Riesz refinement property.*

Proposition 6.2 (Refinement-monoid relation)**.** *For $as, bs, cs, ds : \mathcal{M}(A)$,*

$$as +\!\!+ bs = cs +\!\!+ ds$$
$$\iff$$
$$\exists_{(xs_1, xs_2, ys_1, ys_2 : \mathcal{M}(A))}.\ (as = xs_1 +\!\!+ xs_2)\ \wedge\ (bs = ys_1 +\!\!+ ys_2)\ \wedge\ (xs_1 +\!\!+ ys_1 = cs) \wedge (xs_2 +\!\!+ ys_2 = ds)\ .$$

Using this identity, we obtain the characterisation below of the equality type for finite multisets of the form $x :: xs$.

Theorem 6.3 (Commutation relation)**.** *For a set A, $a, b : A$, and $as, bs : \mathcal{M}(A)$,*

$$a :: as = b :: bs \ \Leftrightarrow\ (a = b \wedge as = bs) \vee (\exists_{(cs : \mathcal{M}(A))}.\ as = b :: cs \wedge a :: cs = bs)\ .$$

The commutation relation can be thought of as consisting of a congruence rule and a *generalised* swap rule. The latter can be written as a deduction rule for multiset equality as follows:

$$\frac{as = b :: cs \qquad a :: cs = bs}{a :: as = b :: bs}\ \mathsf{comm}$$

As exemplified below, the comm rule allows us to generate and compose swap operations.

Example 6.1. *For $a, b, c : A$ and $cs, ds : \mathcal{M}(A)$,*

(i)

$$\dfrac{\dfrac{}{b :: cs = b :: cs}\ \mathsf{refl} \qquad \dfrac{}{a :: cs = a :: cs}\ \mathsf{refl}}{a :: b :: cs = b :: a :: cs}\ \mathsf{comm}$$

(ii)

$$\dfrac{\dfrac{\dfrac{}{b :: ds = b :: ds}\ \mathsf{refl} \qquad \dfrac{}{c :: ds = c :: ds}\ \mathsf{refl}}{b :: c :: ds = c :: b :: ds}\ \mathsf{comm} \qquad \dfrac{\dfrac{}{a :: ds = a :: ds}\ \mathsf{refl} \qquad \dfrac{}{b :: ds = b :: ds}\ \mathsf{refl}}{a :: b :: ds = b :: a :: ds}\ \mathsf{comm}}{a :: b :: c :: ds = c :: b :: a :: ds}\ \mathsf{comm}$$

To show that the commutation relation generates the path space of finite multisets, we define an inductive relation on $\mathcal{M}(A)$ and show an equivalence with the equality type.

Definition 6.4. *For a type A, let $\sim_A : \mathcal{M}(A) \to \mathcal{M}(A) \to \mathcal{U}$ be the inductive relation with the following constructors:*

$$\begin{aligned}
\mathsf{nil\text{-}cong} &: \mathsf{nil} \sim_A \mathsf{nil} \\
\mathsf{cons\text{-}cong} &: \{a\ b : A\}\ \{as\ bs : \mathcal{M}(A)\} \to (a = b) \to (as \sim_A bs) \to (a :: as \sim_A b :: bs) \\
\mathsf{comm\text{-}rule} &: \{a\ b : A\}\ \{as\ bs\ cs : \mathcal{M}(A)\} \to (as \sim_A b :: cs) \to (a :: cs \sim_A bs) \to (a :: as \sim_A b :: bs)
\end{aligned}$$

Note that this relation is not prop-valued, since there may be multiple derivations of the same permutation.

Example 6.2. *For $a : A$, the type $(\ a :: a :: \mathsf{nil} \sim_A a :: a :: \mathsf{nil}\ )$ is inhabited by both of these terms:* $\mathsf{cons\text{-}cong}(\mathsf{refl}, \mathsf{cons\text{-}cong}(\mathsf{refl}, \mathsf{nil\text{-}cong}))$ *and* $\mathsf{comm\text{-}rule}(\ \mathsf{cons\text{-}cong}(\mathsf{refl}, \mathsf{nil\text{-}cong})\ ,\ \mathsf{cons\text{-}cong}(\mathsf{refl}, \mathsf{nil\text{-}cong})\ )$.

Hence, we propositionally truncate the relation. A usual encode-decode argument shows that this is equivalent to the path space of $\mathcal{M}(A)$.

Proposition 6.5. *For a set A and $as, bs : \mathcal{M}(A)$, we have a soundness map $(as \sim_A bs) \to (as =_{\mathcal{M}(A)} bs)$ and a reflexivity map $(\ as : \mathcal{M}(A)\ ) \to \|as \sim_A as\|_{-1}$.*

Theorem 6.6. *For a set A and $as, bs : \mathcal{M}(A)$, we have the characterisation $(\ as =_{\mathcal{M}(A)} bs\ ) \Leftrightarrow \|as \sim_A bs\|_{-1}$.*

6.2 Multiset-equality deduction system

The commutation relation of the previous section can be simply defined as a relation on lists and, in the presence of congruence rules, this yields that two lists are related iff they are equal when regarded as multisets. This provides a new construction of $\mathcal{M}(A)$ as a set-quotient of $\mathcal{L}(A)$, which we present next.

Definition 6.7. *For a type A, let $\sim_A : \mathcal{L}(A) \to \mathcal{L}(A) \to \mathcal{U}$ be the inductive relation generated by the following constructors:*

$$\begin{aligned}
\mathsf{nil\text{-}cong} &: \mathsf{nil} \sim_A \mathsf{nil} \\
\mathsf{cons\text{-}cong} &: \{a\ b : A\}\ \{as\ bs : \mathcal{L}(A)\} \to (a = b) \to (as \sim_A bs) \to (a :: as \sim_A b :: bs) \\
\mathsf{comm\text{-}rule} &: \{a\ b : A\}\ \{as\ bs\ cs : \mathcal{L}(A)\} \to (as \sim_A b :: cs) \to (a :: cs \sim_A bs) \to (a :: as \sim_A b :: bs)
\end{aligned}$$

It is straightforward to show that the relation $\sim_A$ is reflexive and symmetric, and a congruence with respect to $+\!\!+$. It is however non-trivial to show that it is transitive.

Note that $\sim_A$ encodes permutations of the elements of the underlying finite-lists, that is, permutations on the finite set of indices of the list. Formally, observe that $\mathcal{L}(A) \simeq \sum_{n:\mathbb{N}} \mathsf{Fin}_n \to A$. We define a new relation $\approx_A$ on this representation of lists as follows, where two lists are related iff there exists a permutation of the indices that shuffles one list to the other.

Definition 6.8. *For a type A, define:*

$$- \approx_A - : ((n : \mathbb{N}) \times (f : \mathsf{Fin_n} \to A)) \to ((m : \mathbb{N}) \times (g : \mathsf{Fin_m} \to A)) \to \mathcal{U}$$

$$(n, f) \approx_A (m, g) :\equiv (\phi : \mathsf{Fin_n} \xrightarrow{\sim} \mathsf{Fin_m}) \times (f = g \circ \phi)$$

Since permutations are simply invertible functions, it is straightforward to establish that this is an equivalence relation. We now show that $\approx_A$ *extensionally* agrees with our $\sim_A$ relation. To this end, we perform a translation between the two relations in the style of an NbE (Normalisation by Evaluation) algorithm.

First, given a deduction tree for $\sim_A$, we compute the permutation it encodes by means of an eval function. Second, given an inhabitant of $\approx_A$ we reify it to a deduction tree by means of a quote function (see Appendix E for details).

Definition 6.9.

$$\mathsf{eval} : as \sim_A bs \to as \approx_A bs \qquad\qquad \mathsf{quote} : as \approx_A bs \to as \sim_A bs$$

Finally, we establish the transitivity of $\sim_A$ by translating two composable trees in $\sim_A$ to a composable pair in $\approx_A$ and, after using the transitivity of $\approx_A$, quoting back the result to a tree.

Lemma 6.10. *For a type A, $\sim_A$ is an equivalence relation on $\mathcal{L}(A)$ and a congruence with respect to $+\!\!+$.*

The relation $\sim_A$ is not prop-valued, since there may be multiple inhabitants that encode the same permutation of the underlying elements (see Example 6.2). We propositionally truncate it, defining $xs \mathrel{\tilde{\sim}_A} ys :\equiv \|xs \sim_A ys\|_{-1}$, before using it in as a quotient. The resulting effective quotient of finite lists satisfies the categorical universal property of free commutative monoids.

Theorem 6.11. *For a set A, the composite $A \to \mathcal{L}(A) \to \mathcal{L}(A)_{/\tilde{\sim}_A}$ is the free commutative monoid on A.*

Corollary 6.12. *For every set A, we have the equivalence $\mathcal{L}(A)_{/\tilde{\sim}_A} \simeq \mathsf{FCM}(A)$.*

6.3 Commuted-list construction

As a final application of the commutation relation, we give a construction of another HIT cList for finite-multisets, that uses the $\mathsf{comm\text{-}rule}$ as a path constructor. This is an example of a conditional path constructor, which corresponds to a quasi-equational (or partial Horn) theory.

Definition 6.13 (cList)**.** *For a type A, the 1-HIT $\mathsf{cList}(A)$ is given by the following point and path constructors:*

$$\begin{aligned}
\mathsf{nil} &: \mathsf{cList}(A) \\
- :: - &: A \times \mathsf{cList}(A) \to \mathsf{cList}(A) \\
\mathsf{comm} &: \{a\ b : A\}\{as\ bs\ cs : \mathsf{cList}(A)\} \\
&\quad \to (as = b :: cs) \to (a :: cs = bs) \\
&\quad \to a :: as = b :: bs \\
\mathsf{trunc} &: \mathsf{isSet}(\mathsf{cList}(A))
\end{aligned}$$

Unsurprisingly, this HIT also satisfies the universal property of free commutative monoids, making it equivalent to the other presentations.

Proposition 6.14. *For every set A, $\mathsf{cList}(A)$ is a commutative monoid.*

Theorem 6.15 (Universal property of cList)**.** *For every set A, $\eta : A \to \mathsf{cList}(A)$ is the free commutative monoid on A.*

7 Discussion

We have described various constructions of free commutative monoids, or finite-multisets using set-truncated HITs and a set-quotient. Our results show that a significant part of the combinatorics of finite-multisets

can be reduced to constructions using the categorical universal property and a few structural properties. Our main application is a constructive formulation of the relational model of differential linear logic, encompassing the theory of creation/annihilation operators associated to the combinatorial Fock-space construction. From this, we obtained a commutation relation that we used to characterise the path space of finite multisets and further provided a deduction system for multiset equality.

The simplest encoding of finite multisets is lists up to permutation of their elements. In this context, the relation of Definition 6.8 using automorphisms on finite cardinals as permutations has been considered by Altenkirch et al. [1] and Nuo [35]; while Danielsson [16] considers a similar relation for bag equivalence of lists. Angiuli et al. [5] consider two representations of finite-multisets in Cubical Agda: one is the swapped-list construction and the other is association lists encoding elements with their multiplicity, showing their equivalence. They are interested in the representation independence of data structures and assume decidable equality on the carrier set.

Multisets in type theory have also been studied by Gylterud [25] from the point of view of constructive foundations, considering how to generalise set-theoretic axioms from sets to multisets. They take the definition of a set as a W-type, and use a different equality relation to obtain multisets.

Basold et al. [6] and Frumin et al. [24] consider Kuratowski-finite sets in HoTT, defined as free join-semilattices using the universal-algebraic construction. Our universal-algebraic construction of free commutative monoids is similar but lacking the idempotence axiom.

The type FCM (Definition 2.8) is a non-recursive HIT, the type sList (Definition 2.10) is a recursive HIT, and the type cList (Definition 6.13) is a recursive HIT with conditional path constructors. These definitions of HITs are accepted by (the current version of) Cubical Agda [40]. However, it is unclear whether the known schemas for HITs in cubical type theories can encode conditional path constructors. Fiore et al. [23] consider sList in their quotient-inductive types framework and mention cList as an example that goes beyond it.

This work is about 0-truncated finite multisets, which only allow elimination into sets. This is restrictive. A preliminary presentation of our work [13] included a proposal for extensions to free symmetric monoidal groupoids and the construction of the bicategory of generalised species for groupoids. In particular, generalising the above HITs to groupoids requires higher path constructors; for example, sList requires a higher path constructor to establish the coherence of swap.

It is folklore that genuine homotopy finite-multi-types in HoTT should be given by the construction $\mathcal{M}(A) :\equiv \sum_{X:\mathcal{U}_{\mathsf{Fin}}} \pi_1(X) \to A$ where $\mathcal{U}_{\mathsf{Fin}} :\equiv \sum_{X:\mathcal{U}} \sum_{n:\mathbb{N}} \|X = \mathsf{Fin}_{\mathsf{n}}\|_{-1}$; see, for instance, [10]. It is generally believed that this should satisfy the universal property of the free symmetric monoidal (∞) groupoid on the (∞) groupoid A, but it is unknown how to prove this internally in HoTT.

Truncated to 1-groupoids, the construction of HITs that satisfy the universal property of free symmetric monoidal groupoids has been considered by Piceghello [37,36]. The equivalence of the free symmetric monoidal groupoid on one generator with $\mathcal{U}_{\mathsf{Fin}}$ in HoTT, has been considered by Choudhury et al. [14] in the context of fully-abstract models for reversible programming languages [14,12]. We believe that it is possible to apply these techniques to generalise the work presented in this paper to groupoids.

References

[1] Altenkirch, T., T. Anberrée and N. Li, *Definable Quotients in Type Theory* , p. 22.

[2] Altenkirch, T., J. Chapman and T. Uustalu, *Monads Need Not Be Endofunctors*, in: L. Ong, editor, *Foundations of Software Science and Computational Structures*, Lecture Notes in Computer Science, pp. 297–311.

[3] Altenkirch, T. and L. Scoccola, *The Integers as a Higher Inductive Type*, in: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, pp. 67–73.
URL https://doi.org/10.1145/3373718.3394760

[4] Angiuli, C., *Computational Semantics of CartesianCubical Type Theory.*
URL https://www.cs.cmu.edu/~cangiuli/thesis/thesis.pdf

[5] Angiuli, C., E. Cavallo, A. Mörtberg and M. Zeuner, *Internalizing representation independence with univalence* **5**, pp. 12:1–12:30.
URL https://doi.org/10.1145/3434293

[6] Basold, H., H. Geuvers and N. van der Weide, *Higher Inductive Types in Programming* .
URL http://www.jucs.org/doi?doi=10.3217/jucs-023-01-0063

[7] Benton, N., G. Bierman, V. de Paiva and M. Hyland, *Linear λ-calculus and categorical models revisited*, in: E. Börger, G. Jäger, H. Kleine Büning, S. Martini and M. M. Richter, editors, *Computer Science Logic*, Lecture Notes in Computer Science, pp. 61–84.

[8] Blute, R. F., J. R. B. Cockett, J.-S. P. Lemay and R. A. G. Seely, *Differential Categories Revisited* **28**, pp. 171–235.
URL https://doi.org/10.1007/s10485-019-09572-y

[9] Blute, R. F., J. R. B. Cockett and R. a. G. Seely, *Differential categories* **16**, pp. 1049–1083.
URL https://www.cambridge.org/core/journals/mathematical-structures-in-computer-science/article/differential-categories/794F222FE0710CBEF06492127AF4E5CF

[10] Buchholtz, U., *Genuine pairs and the trouble with triples in homotopy type theory* , p. 28.
URL https://types21.liacs.nl/timetable/event/genuine-pairs-and-the-trouble-with-triples-in-homotopy-type-theory/

[11] Buchholtz, U., F. van Doorn and E. Rijke, *Higher Groups in Homotopy Type Theory*, in: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pp. 205–214.
URL https://doi.org/10.1145/3209108.3209150

[12] Carette, J., C.-H. Chen, V. Choudhury and A. Sabry, *From Reversible Programs to Univalent Universes and Back* **336**, pp. 5–25.
URL https://linkinghub.elsevier.com/retrieve/pii/S1571066118300161

[13] Choudhury, V. and M. Fiore, *The finite-multiset construction in HoTT* , p. 40.
URL https://hott.github.io/HoTT-2019/conf-slides/Choudhury.pdf

[14] Choudhury, V., J. Karwowski and A. Sabry, *Symmetries in reversible programming: From symmetric rig groupoids to reversible programming languages* **6**, pp. 6:1–6:32.
URL https://doi.org/10.1145/3498667

[15] Cohen, C., T. Coquand, S. Huber and A. Mörtberg, *Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom* , p. 34 pages.
URL http://drops.dagstuhl.de/opus/volltexte/2018/8475/

[16] Danielsson, N. A., *Bag Equivalence via a Proof-Relevant Membership Relation*, in: L. Beringer and A. Felty, editors, *Interactive Theorem Proving*, Lecture Notes in Computer Science, pp. 149–165.

[17] Day, B., *On closed categories of functors*, in: S. MacLane, H. Applegate, M. Barr, B. Day, E. Dubuc, Phreilambud, A. Pultr, R. Street, M. Tierney and S. Swierczkowski, editors, *Reports of the Midwest Category Seminar IV*, Lecture Notes in Mathematics, pp. 1–38.

[18] Day, B. J., *Note on compact closed categories* **24**, pp. 309–311.
URL https://www.cambridge.org/core/journals/journal-of-the-australian-mathematical-society/article/note-on-compact-closed-categories/F36D383A92F41665ED4BBE9F8F199B20

[19] Ehrhard, T., *An introduction to differential linear logic: Proof-nets, models and antiderivatives* **28**, pp. 995–1060.
URL https://www.cambridge.org/core/journals/mathematical-structures-in-computer-science/article/an-introduction-to-differential-linear-logic-proofnets-models-and-antiderivatives/9852C5F1762B016666DD8DE35129C534

[20] Fiore, M., *An Axiomatics and a Combinatorial Model of Creation/Annihilation Operators*.
URL http://arxiv.org/abs/1506.06402

[21] Fiore, M., N. Gambino, M. Hyland and G. Winskel, *Relative pseudomonads, Kleisli bicategories, and substitution monoidal structures* **24**, pp. 2791–2830.
URL https://doi.org/10.1007/s00029-017-0361-3

[22] Fiore, M. P., *Differential Structure in Models of Multiplicative Biadditive Intuitionistic Linear Logic*, in: S. R. Della Rocca, editor, *Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science, pp. 163–177.

[23] Fiore, M. P., A. M. Pitts and S. C. Steenkamp, *Quotients, inductive types, and quotient inductive types*.
URL http://arxiv.org/abs/2101.02994

[24] Frumin, D., H. Geuvers, L. Gondelman and N. van der Weide, *Finite sets in homotopy type theory*, in: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2018, pp. 201–214.
URL https://doi.org/10.1145/3167085

[25] Gylterud, H. k. R., *Multisets in type theory* **169**, pp. 1–18.
URL https://www.cambridge.org/core/journals/mathematical-proceedings-of-the-cambridge-philosophical-society/article/abs/multisets-in-type-theory/C17604B9927E477B70126529A2A91321

[26] Hyland, J. M. E., *Elements of a theory of algebraic theories* **546**, pp. 132–144.
URL https://doi.org/10.1016/j.tcs.2014.03.005

[27] Hyland, M., *Some reasons for generalising domain theory* **20**, pp. 239–265.
URL https://www.cambridge.org/core/product/identifier/S0960129509990375/type/journal_article

[28] Hyland, M. and A. Schalk, *Glueing and orthogonality for models of linear logic* **294**, pp. 183–231.
URL https://www.sciencedirect.com/science/article/pii/S0304397501002419

[29] Kapulkin, C., P. L. Lumsdaine and V. Voevodsky, *Univalence in Simplicial Sets*.
URL http://arxiv.org/abs/1203.2553

[30] Kelly, G. M. and M. L. Laplaza, *Coherence for compact closed categories* **19**, pp. 193–213.
URL https://www.sciencedirect.com/science/article/pii/0022404980901012

[31] Kock, A., *Monads on symmetric monoidal closed categories* **21**, pp. 1–10.
URL https://doi.org/10.1007/BF01220868

[32] Kraus, N. and T. Altenkirch, *Free Higher Groups in Homotopy Type Theory*, in: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pp. 599–608.
URL https://doi.org/10.1145/3209108.3209183

[33] Lafont, Y., *The linear abstract machine* **59**, pp. 157–180.
URL https://www.sciencedirect.com/science/article/pii/0304397588901004

[34] Lawvere, F. W., *Metric spaces, generalized logic, and closed categories* **43**, pp. 135–166.
URL https://doi.org/10.1007/BF02924844

[35] Li, N., *Quotient types in type theory*.
URL http://eprints.nottingham.ac.uk/28941/

[36] Piceghello, S., "Coherence for Monoidal and Symmetric Monoidal Groupoids in Homotopy Type Theory," The University of Bergen.
URL https://bora.uib.no/bora-xmlui/handle/11250/2830640

[37] Piceghello, S., *Coherence for symmetric monoidal groupoids in HoTT/UF* , p. 2.
URL http://www.ii.uib.no/~bezem/abstracts/TYPES_2019_paper_10

[38] Seely, R. A. G., *Linear Logic, *-Autonomous Categories and Cofree Coalgebras*, in: *In Categories in Computer Science and Logic*, pp. 371–382.

[39] Univalent Foundations Program, T., "Homotopy Type Theory: Univalent Foundations of Mathematics," Univalent Foundations Program.
URL https://homotopytypetheory.org/book

[40] Vezzosi, A., A. Mörtberg and A. Abel, *Cubical agda: A dependently typed programming language with univalence and higher inductive types* **3**, pp. 87:1–87:29.
URL https://doi.org/10.1145/3341691

[41] Voevodsky, V., *Resizing Rules - their use and semantic justification*.
URL https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/2011_Bergen.pdf

A Supplementary material for Section 2

Proposition 2.6. *For all sets A and B, $\tau_{A,B} \circ \sigma_{A,\mathcal{M}(B)}{}^\sharp = \sigma_{A,B}{}^\sharp \circ \tau_{\mathcal{M}(A),B} : \mathcal{M}(A) \times \mathcal{M}(B) \to \mathcal{M}(A \times B)$. This map is a homomorphism in each argument and universal amongst all such.*

Proof.

$$\begin{array}{ccc} \mathcal{M}(A) \times \mathcal{M}(B) & \xrightarrow{\sigma_{A,\mathcal{M}(B)}} & \mathcal{M}(A \times \mathcal{M}(B)) \\ \downarrow{\scriptstyle \tau_{\mathcal{M}(A),B}} & & \downarrow{\scriptstyle \tau_{A,B}{}^\sharp} \\ \mathcal{M}(\mathcal{M}(A) \times B) & \xrightarrow[\sigma_{A,B}{}^\sharp]{} & \mathcal{M}(A \times B) \end{array}$$

We need to establish the equality $\tau^\sharp \circ \sigma = \sigma^\sharp \circ \tau$. By function extensionality, we need to show that for all $as : \mathsf{sList}(A)$ and $bs : \mathsf{sList}(B)$, $\tau^\sharp(\sigma(as, bs)) = \sigma^\sharp(\tau(as, bs))$. This follows by induction on as and bs, using the propositional induction principle. □

Proposition 2.7. *The free commutative-monoid monad is canonically a strong symmetric monoidal endofunctor from the coproduct monoidal structure to the product monoidal structure; the monoidal isomorphisms are*

$$\begin{array}{ccccc} \mathcal{M}(A) \times \mathcal{M}(B) & & \xrightarrow{\simeq} & & \mathcal{M}(A+B) \\ & {\scriptstyle \mathcal{M}(\iota_1) \times \mathcal{M}(\iota_2)} \searrow & & \nearrow {\scriptstyle \mathcal{M}(A+B) \times \mathcal{M}(A+B)} & \\ & & \mathcal{M}(A+B) \times \mathcal{M}(A+B) & & \end{array} \quad , \quad \mathbf{1} \xrightarrow[\mathsf{nil}]{\simeq} \mathcal{M}(\mathbf{0})$$

Proof. $\mathcal{M}(A) \times \mathcal{M}(B)$ is a commutative monoid using the cartesian product, and we get a homomorphism $\mathcal{M}(A + B) \to \mathcal{M}(A) \times \mathcal{M}(B)$ by extending the map $(\eta_A(-) \times \mathsf{nil}, \mathsf{nil} \times \eta_B(-)) : A + B \to \mathcal{M}(A) \times \mathcal{M}(B)$. The map $\mathcal{M}(A) \times \mathcal{M}(B) \to \mathcal{M}(A + B)$ is bilinear, that is, linear in each argument, and we use the uniqueness of the universal property to establish the equivalence. □

Definition A.1 (Induction principle for FCM)**.** *Given a type family $P : \mathsf{FCM}(A) \to \mathcal{U}$ with the following data:*

$$\begin{aligned} \eta^* &: (x : A) \to P(\eta(x)) \\ e^* &: P(e) \\ - \cdot^* - &: \{x\ y : \mathsf{FCM}(A)\} \to P(x) \to P(y) \to P(x \cdot y) \\ \mathsf{assoc}^* &: \{x\ y\ z : \mathsf{FCM}(A)\}(p : P(x))(q : P(y))(r : P(z)) \\ &\quad \to p \cdot^* (q \cdot^* r) =^P_{\mathsf{assoc}(x,y,z)} (p \cdot^* q) \cdot^* r \\ \mathsf{unitl}^* &: \{x : \mathsf{FCM}(A)\}(p : P(x)) \to e^* \cdot^* p =^P_{\mathsf{unitl}(x)} p \\ \mathsf{unitr}^* &: \{x : \mathsf{FCM}(A)\}(p : P(x)) \to p \cdot^* e^* =^P_{\mathsf{unitr}(x)} p \\ \mathsf{comm}^* &: \{x\ y : \mathsf{FCM}(A)\}(p : P(x))(q : P(y)) \\ &\quad \to p \cdot^* q =^P_{\mathsf{comm}(x,y)} q \cdot^* p \\ \mathsf{trunc}^* &: (x : \mathsf{FCM}(A)) \to \mathsf{isSet}(P(x)) \end{aligned}$$

there is a unique function $f : (x : \mathsf{FCM}(A)) \to P(x)$ with the following computation rules:

$$
\begin{aligned}
f(\eta(x)) &\equiv \eta^*(x) \\
f(e) &\equiv e^* \\
f(x \cdot y) &\equiv f(x) \cdot^* f(y) \\
\mathsf{apd}_f(\mathsf{assoc}(x,y,z)) &= \mathsf{assoc}^*(f(x), f(y), f(z)) \\
\mathsf{apd}_f(\mathsf{unitl}(x)) &= \mathsf{unitl}^*(f(x)) \\
\mathsf{apd}_f(\mathsf{unitr}(x)) &= \mathsf{unitr}^*(f(x)) \\
\mathsf{apd}_f(\mathsf{comm}(x,y)) &= \mathsf{comm}^*(f(x), f(y))
\end{aligned}
$$

Note that we use judgemental equality for the computation rules for the point constructors, and the computation rules for the path constructors hold up to the identity type.

Theorem 2.9. *For every set A, $\eta : A \to \mathsf{FCM}(A)$ is the free commutative monoid on A.*

Proof. Assuming $f : A \to M$, we define $f^\sharp : \mathsf{FCM}(A) \to M$ by induction. It is straightforward to check that $f^\sharp$ is a homomorphism. Finally, we need show that $(-)^\sharp$ is inverse to $(-) \circ \eta$. We establish the homotopies $(h \circ \eta)^\sharp \sim h$ and $f^\sharp \circ \eta \sim f$ by calculation. □

Definition A.2 (Induction principle for sList)**.**
Given a type family $P : \mathsf{sList}(A) \to \mathcal{U}$ with the following data:

$$
\begin{aligned}
&\mathsf{nil}^* : P(\mathsf{nil}) \\
&\_ ::^* \_ : (x : A)\{xs : \mathsf{sList}(A)\} \to P(xs) \to P(x :: xs) \\
&\mathsf{swap}^* : (x\ y : A)\{xs : \mathsf{sList}(A)\}(p : P(xs)) \to\ x ::^* y ::^* p =^P_{\mathsf{swap}(x,y,xs)} y ::^* x ::^* p \\
&\mathsf{trunc}^* : (xs : \mathsf{sList}(A)) \to \mathsf{isSet}(P(xs))
\end{aligned}
$$

there is a unique function $f : (xs : \mathsf{sList}(A)) \to P(xs)$ satisfying the following computation rules:

$$
\begin{aligned}
f(\mathsf{nil}) &\equiv nil* \\
f(x :: xs) &\equiv x ::^* f(xs) \\
\mathsf{apd}_f(\mathsf{swap}(x,y,xs)) &= \mathsf{swap}^*(x,y,f(xs))
\end{aligned}
$$

Lemma 2.11 (Propositional induction principle for sList)**.**
Let $P : \mathsf{sList}(A) \to \mathcal{U}$ be a type family with the following data.

$$
\begin{aligned}
&\mathsf{nil}^* : P(\mathsf{nil}) \\
&\_ ::^* \_ : (x : A)\{xs : \mathsf{sList}(A)\} \to P(xs) \to P(x :: xs) \\
&\mathsf{trunc}^* : (xs : \mathsf{sList}(A)) \to \mathsf{isProp}(P(xs))
\end{aligned}
$$

Then, there is a unique function $f : (xs : \mathsf{sList}(A)) \to P(xs)$ satisfying appropriate computation rules.

Proof. Using the standard induction principle for sList, we need to give an image swap^* for swap. Consider $\phi_1 : P(x :: y :: xs)$ and $\phi_2 : P(y :: x :: xs)$. Transporting ϕ_1 along swap, and using the fact that they are propositions, one gets the path $\phi_1 =^P_{\mathsf{swap}(x,y,xs)} \phi_2$. □

Lemma 2.14. *The concatenation operation $+\!\!+$ is associative and nil is a left and right unit. For all $xs, ys, zs : \mathsf{sList}(A)$, we have $xs +\!\!+ (ys +\!\!+ zs) = (xs +\!\!+ ys) +\!\!+ zs$, $\mathsf{nil} +\!\!+ xs = xs$, and $xs +\!\!+ \mathsf{nil} = xs$.*

Proof. By the propositional induction principle, the only required cases are for nil and $::$, and this is the same proof as for lists. □

Lemma 2.15. *For any $x : A$, and $xs, ys : \mathsf{sList}(A)$, we have $x :: xs = xs +\!\!+ [x]$, and $xs +\!\!+ ys = ys +\!\!+ xs$.*

Proof. The idea is to recursively apply the swap constructor to shift the first element to the end. To prove commutativity by induction, the multiset is first split into appends of singleton multisets.

(i) By induction on xs, we have

$$x :: \mathsf{nil} = \mathsf{nil} +\!\!+ [x] \qquad \text{left unit}$$

and

$$\begin{aligned} & x :: (y :: xs) & \\ =\ & y :: (x :: xs) & \text{by } \mathsf{swap}(x, y, xs) \\ =\ & y :: (xs +\!\!+ [x]) & \text{induction hypothesis} \\ \equiv\ & (y :: xs) +\!\!+ [x] & \text{definition} \end{aligned}$$

(ii) By induction on xs, we have

$$\begin{aligned} & \mathsf{nil} +\!\!+ ys & \\ \equiv\ & ys & \text{definition} \\ =\ & ys +\!\!+ \mathsf{nil} & \text{right unit} \end{aligned}$$

and

$$\begin{aligned} & (x :: xs) +\!\!+ ys & \\ \equiv\ & x :: (xs +\!\!+ ys) & \text{definition} \\ =\ & x :: (ys +\!\!+ xs) & \text{induction hypothesis} \\ \equiv\ & (x :: ys) +\!\!+ xs & \text{definition} \\ =\ & (ys +\!\!+ [x]) +\!\!+ xs & \text{by (i)} \\ =\ & ys +\!\!+ ([x] +\!\!+ xs) & \text{associativity} \\ \equiv\ & ys +\!\!+ (x :: xs) & \text{definition} \end{aligned}$$

□

Proposition 2.17. *Given a set A, a commutative monoid M, and a map $f : A \to M$ of sets, f extends to a homomorphism $f^\sharp : \mathsf{CMon}(\mathsf{sList}(A), M)$.*

Proof. We define $f^\sharp : \mathsf{sList}(A) \to M$ by induction. On the point constructors, we define

$$\begin{aligned} f^\sharp(\mathsf{nil}) &:\equiv e \\ f^\sharp(x :: xs) &:\equiv f(x) \cdot f^\sharp(xs) \end{aligned}$$

On the path constructor, we have

$$\begin{aligned} & f(x) \cdot f(y) \cdot f^\sharp(xs) & \\ =\ & (f(x) \cdot f(y)) \cdot f^\sharp(xs) & \text{associativity} \\ =\ & (f(y) \cdot f(x)) \cdot f^\sharp(xs) & \text{commutativity} \\ =\ & f(y) \cdot f(x) \cdot f^\sharp(xs) & \text{associativity} \end{aligned}$$

To check that this is a homomorphism, we establish that $f^\sharp(xs +\!\!+ ys) = f^\sharp(xs) \cdot f^\sharp(ys)$ for all $xs, ys : \mathsf{sList}(A)$. Since this is a proposition, by induction on xs, we have

$$\begin{aligned} & f^\sharp(\mathsf{nil} +\!\!+ ys) & \\ \equiv\ & f^\sharp(ys) & \text{definition} \\ =\ & e \cdot f^\sharp(ys) & \text{unit law} \\ \equiv\ & f^\sharp(\mathsf{nil}) \cdot f^\sharp(ys) & \text{definition} \end{aligned}$$

$$
\begin{aligned}
& f^\sharp((x :: xs) +\!\!+ ys) \\
\equiv\ & f^\sharp(x :: (xs +\!\!+ ys)) && \text{definition} \\
\equiv\ & f(x) \cdot f^\sharp(xs +\!\!+ ys) && \text{definition} \\
=\ & f(x) \cdot (f^\sharp(xs) \cdot f^\sharp(ys)) && \text{induction hypothesis} \\
=\ & (f(x) \cdot f^\sharp(xs)) \cdot f^\sharp(ys) && \text{associativity} \\
\equiv\ & f^\sharp(x :: xs) \cdot f^\sharp(ys) && \text{definition}
\end{aligned}
$$

□

Theorem 2.18. *For every set A, $\eta : A \to \mathsf{sList}(A)$ is the free commutative monoid on A.*

Proof. We need to show that $(-)^\sharp$ is inverse to $(-) \circ \eta$. We establish the homotopies $(h \circ \eta)^\sharp \sim h$ and $f^\sharp \circ \eta \sim f$ by calculation. □

Corollary 2.19. *For every set A, we have an equivalence $\mathsf{sList}(A) \simeq \mathsf{FCM}(A)$.*

Proof. The two maps are constructed by extending η, that is, $\eta^\sharp$. Using the universal property, their composition is homotopic to id. □

B Supplementary material for Section 3

Proposition 3.3. *The following identities hold.*

(i) ${\text{よ}_A}^* = \mathsf{id}_{\mathfrak{P}(A)}$.

(ii) *For $f : A \to \mathfrak{P}(B)$, we have $f = f^* \circ \text{よ}_A$.*

(iii) *For $f : A \to \mathfrak{P}(B)$ and $g : B \to \mathfrak{P}(C)$, we have that $(g^* \circ f)^* = g^* \circ f^*$.*

Proof.

(i) For any $f : A \to \mathfrak{P}(A)$ and $a : A$, we have

$$
\begin{aligned}
{\text{よ}_A}^*(f, a) &\equiv \exists_{(b:A)}.\text{よ}_A(a, b) \times f(a) \\
&\equiv \exists_{(b:A)}.(a = b) \times f(a) \\
&\simeq f(a)
\end{aligned}
$$

(ii) For any $a : A$ and $b : B$, we have

$$
\begin{aligned}
f^*(\text{よ}_A(a), b) &\equiv \exists_{(c:A)}.f(c, b) \times \text{よ}_A(a, c) \\
&\equiv \exists_{(c:A)}.f(c, b) \times (a = c) \\
&= f(a, b)
\end{aligned}
$$

(iii) For any $\alpha : \mathfrak{P}(A)$ and $c : C$, we have

$$
\begin{aligned}
& (g^* \circ f)^*(\alpha, c) \\
\equiv\ & \exists_{(x:A)}.(g^* \circ f)(x, c) \times \alpha(x) \\
\equiv\ & \exists_{(x:A)}.g^*(f(x))(c) \times \alpha(x) \\
\equiv\ & \exists_{(x:A)}.\exists_{(y:B)}.(g(y, c) \times f(x, y)) \times \alpha(x) \\
\simeq\ & \exists_{(y:B)}.g(y, c) \times (\exists_{(x:A)}.f(x, y) \times \alpha(x)) \\
\equiv\ & \exists_{(y:B)}.(g(y, c) \times f^*(\alpha)(y)) \\
\equiv\ & g^*(f^*(\alpha))(c) \\
\equiv\ & (g^* \circ f^*)(\alpha, c)
\end{aligned}
$$

□

Proposition 3.5. Rel *is a (univalent) category.*

Proof. We note that $\mathsf{Hom}_{\mathsf{Rel}}(A, B) :\equiv A \nrightarrow B$ is an hSet since hProp is. The identity arrow $1_A : \mathsf{Hom}(A, A)$ is given by $\lambda\!\!\!^{\dagger}_A$. The composition of $g : B \nrightarrow C$ and $f : A \nrightarrow B$ is given by $g^* \circ f$. The unit and associativity laws follow from Proposition 3.3. To show that Rel is univalent, see [39, Example 9.1.19]. □

C Supplementary material for Section 4

Theorem 4.2. *For every set A, the set $\mathcal{M}(A)$ with multiplication $m :\equiv (+\!\!+_A)_* : \mathcal{M}(A) \otimes \mathcal{M}(A) \nrightarrow \mathcal{M}(A)$ and unit $e :\equiv (\lambda(x:\mathbf{1}).\,\mathsf{nil})_* : \mathbf{1} \nrightarrow \mathcal{M}(A)$ equipped with $(\eta_A)_* : A \nrightarrow \mathcal{M}(A)$ is the free commutative monoid on A in* Rel.

Proof. For a commutative monoid M in Rel and $f : A \nrightarrow M$, one needs to define a unique homomorphic extension $\mathcal{M}(A) \nrightarrow M$. Using that $\mathfrak{P}(M)$ is a commutative monoid in Set, this is given by $f^\sharp : \mathcal{M}(A) \to \mathfrak{P}(M)$. The rest of the argument is established by calculation. □

Corollary 4.5. *For every set A, $\epsilon_A :\equiv ((\eta_A)_*)^\dagger : \mathcal{M}(A) \nrightarrow A$ is the cofree commutative comonoid on A in* Rel*; that is, for a commutative comonoid K, every relation $r : K \nrightarrow A$ has a unique homomorphic coextension $r_\sharp : K \nrightarrow \mathcal{M}(A)$ over ϵ_A.*

Proof. This follows from Theorem 4.2 by self duality of Rel. For a commutative comonoid (K, w, k) in Rel and $r : K \nrightarrow A$, the homomorphic coextension $r_\sharp : K \nrightarrow \mathcal{M}(A)$ is $((r^\dagger)^\sharp)^\dagger$ where the homomorphic extension is taken with respect to the promonoidal convolution on the commutative monoid $(K, w^\dagger, e^\dagger)$. □

D Supplementary material for Section 5

Proposition 5.4. *Let A be a set.*

(i) *There is a unique choice function* $\mathsf{uc} : (as : \mathcal{M}(A)) \to (\ell(as) = 1) \to \mathsf{isSing}(as)$.

(ii) *The universal map $\eta_A : A \to \mathcal{M}(A)$ is an embedding; that is,* $\mathsf{ap}_{\eta_A} : x = y \to [x] = [y]$ *is an equivalence.*

(iii) *For $as : \mathcal{M}(A)$,* $\mathsf{isSing}(as)$ *is a proposition.*

(iv) *For $as : \mathcal{M}(A)$, we have that* $\mathsf{uc}(as)$ *is an equivalence.*

Proof. (i) The case for nil is eliminated since it has length 0. Knowing that $a :: as$ has length 1 implies that as has length 0 and is hence empty. Finally, swap only acts on multisets of length at least 2, hence the case for swap gets eliminated.

(ii) Since A and $\mathcal{M}(A)$ are sets, we only need to show that η is injective, that is, $[a] = [b]$ implies $a = b$. Since these are length-one multisets, we simply apply the uc function.

(iii) This follows from η being injective.

(iv) Since they are both propositions, we simply need an inverse map to uc, which we get by applying ℓ.

□

Proposition 5.5. *For a set A,*

$$A \;\simeq\; \textstyle\sum_{as:\mathcal{M}(A)} (\ell(as) = 1) \;\simeq\; \sum_{as:\mathcal{M}(A)} \mathsf{isSing}(as) \;\;.$$

Proof. This follows from Proposition 5.4 using the unique choice function. □

Proposition 5.6. *Let A be a set.*

(i) *For $s : \mathcal{M}(\mathcal{M}(A))$ and $a : A$, $\mu(s) = [a] \Leftrightarrow \exists_{(t:\mathcal{M}(\mathcal{M}(A)))}.\mu(t) = \mathsf{nil} \wedge [a] :: t = s$.*

(ii) *For a set B and $t : \mathcal{M}(A \times B)$, $a : A$, $\mathcal{M}(\pi_1)(t) = [a] \;\Leftrightarrow\; \exists_{(b:B)}.t = [(a, b)]$.*

(iii) *For $as, bs : \mathcal{M}(A)$ and $a : A$, $as +\!\!+ bs = [a] \Leftrightarrow (as = [a] \wedge bs = \mathsf{nil}) \vee (as = \mathsf{nil} \wedge bs = [a])$.*

Proof. Since these are equivalences of propositions, one simply needs to write down both the maps. Going from right to left is easy and just follows by calculation. To go from left to right, we use the characterisation of the path space of singleton multisets and the unique choice function from Proposition 5.4. □

E Supplementary material for Section 6

Theorem 6.3 (Commutation relation)**.** *For a set A, $a, b : A$, and $as, bs : \mathcal{M}(A)$,*

$$a :: as = b :: bs \;\Leftrightarrow\; (a = b \wedge as = bs) \vee (\exists_{(cs:\mathcal{M}(A))}.\ as = b :: cs \wedge a :: cs = bs) \ .$$

Proof. This follows from the bialgebra law (refinement-monoid relation) and using the following sequence of manipulations which use the characterisation of singletons.

(i) For $a, b : A$, and $as : \mathcal{M}(A)$, we have

$$a :: as = [b] \Longleftrightarrow (a = b) \wedge (as = \mathsf{nil}) \ .$$

(ii) For $a, b : A$, and $as, cs : \mathcal{M}(A)$, we have

$$\exists_{(xs:\mathcal{M}(A))}.(a :: xs = [b]) \wedge (as = xs +\!\!+ cs) \Longleftrightarrow (a = b) \wedge (as = cs) \ .$$

(iii) For $a : A$, and $as, bs, cs : \mathcal{M}(A)$, we have

$$(a :: as) = (bs +\!\!+ cs)$$
$$\Longleftrightarrow$$
$$\exists_{(xs:\mathcal{M}(A))}.(a :: xs = bs) \wedge (as = xs +\!\!+ cs) \vee \exists_{(ys:\mathcal{M}(A))}.(as = bs +\!\!+ ys) \wedge (a :: ys = cs) \ .$$

□

To reason about and remove intermediate elements in Fin_n, we define a type family $\mathsf{Fin}^-_\mathsf{n}$ over Fin_n, which picks out the set of elements in Fin_n except a chosen one (see [14, Section 5.6]).

Definition E.1. $\mathsf{Fin}^-_\mathsf{n}(k) :\equiv \sum_{i:\mathsf{Fin}_\mathsf{n}} i \neq k$

We observe the following identities, which let us lift equivalences on Fin_n to equivalences on $\mathsf{Fin}^-_\mathsf{n}$.

Proposition E.2.

(i) *For any $i : \mathsf{Fin}_{\mathsf{S}\,\mathsf{n}}$, we have $\mathsf{Fin}^-_{\mathsf{S}\,\mathsf{n}}(i) \simeq \mathsf{Fin}_\mathsf{n}$.*

(ii) *Given an equivalence $\phi : \mathsf{Fin}_{\mathsf{S}\,\mathsf{n}} \simeq \mathsf{Fin}_{\mathsf{S}\,\mathsf{m}}$ and $i : \mathsf{Fin}_{\mathsf{S}\,\mathsf{n}}$, we have $\mathsf{Fin}^-_{\mathsf{S}\,\mathsf{n}}(i) \simeq \mathsf{Fin}^-_{\mathsf{S}\,\mathsf{m}}(\phi(i))$.*

We can increment or decrement permutations, by the following. To increment a permutation by one, we add to the beginning, mapping 0 to 0. To decrement a permutation, we remove a given element i and its image $\phi(i)$.

Proposition E.3.

(i) *Given an equivalence $\phi : \mathsf{Fin}_\mathsf{n} \simeq \mathsf{Fin}_\mathsf{m}$, we have an equivalence $\mathsf{Fin}_{\mathsf{S}\,\mathsf{n}} \simeq \mathsf{Fin}_{\mathsf{S}\,\mathsf{m}}$.*

(ii) *Given an equivalence $\phi : \mathsf{Fin}_{\mathsf{S}\,\mathsf{n}} \simeq \mathsf{Fin}_{\mathsf{S}\,\mathsf{m}}$ and $i : \mathsf{Fin}_{\mathsf{S}\,\mathsf{n}}$, we have an equivalence $\mathsf{Fin}_\mathsf{n} \simeq \mathsf{Fin}_\mathsf{m}$.*

Definition 6.9.

$$\mathsf{eval} : as \sim_A bs \to as \approx_A bs \qquad\qquad \mathsf{quote} : as \approx_A bs \to as \sim_A bs$$

Proof. We define eval by induction on $as \sim_A bs$. The cases for nil-cong and cons-cong are trivial. For the case comm-rule, we need to calculate $a :: as \approx_A b :: bs$. We recursively calculate $as \approx_A b :: cs$ and $a :: cs \approx_A bs$, and then, by congruence, we have $a :: as \approx_A a :: b :: cs$ and $b :: a :: cs \approx_A b :: bs$. We construct the permutation that swaps the first two elements to get $a :: b :: cs \approx_A b :: a :: cs$. Finally, we compose the three to get $a :: as \approx_A b :: bs$.

To construct quote, we start with $as \approx_A bs$, that is, we have a $\phi : \mathsf{Fin}_n \simeq \mathsf{Fin}_m$, such that, $as = bs \circ \phi$. Knowing ϕ, we deduce that $n = m$, and then we do an induction on n.

Case 0: Follows by nil-cong.

Case $\mathsf{S}\,0$: Since $\mathsf{Fin}_1 \simeq \mathsf{Fin}_1$ is contractible, ϕ is the identity. Using this, we get a path between the head elements $as(0) = bs(0)$, and then apply cons-cong using nil-cong.

Case $\mathsf{S}\,\mathsf{S}\,n$: We use the decidable equality on $\mathsf{Fin}_{\mathsf{S}\,\mathsf{S}\,n}$ to test if $\phi(0) = 0$.

Case $\phi(0) = 0$: We get a path between the head elements $as(0) = bs(\phi(0)) = bs(0)$. We recursively compute on the rest of the list, after deleting 0 and $\phi(0)$ from the permutation ϕ by Proposition E.3. Finally, we combine them using cons-cong.

Case $\phi(0) \neq 0$: In this case, we use comm-rule. We delete the elements at position 0 and $\phi(0)$ in bs and choose this to be cs. For the first argument to comm-rule, we recursively compute using the lists: f with the element at 0 removed, and $g(0)$ appended to cs. For the second argument, we recursively compute using the lists: $f(0)$ appended to cs, and g with the element at 0 removed. For both these recursive calls, we have to produce a compatible permutation, and this is given by deleting 0 and $\phi(0)$ from ϕ using Proposition E.3.

□

Lemma 6.10. *For a type A, $\sim_A$ is an equivalence relation on $\mathcal{L}(A)$ and a congruence with respect to $+\!\!+$.*

Proof. Transitivity follows by Definition 6.9 and using the transitivity of the relation $\approx_A$. The rest is straightforward to establish. □

Definition E.4 (Induction principle for cList)**.** *For every type family $P : \mathsf{cList}(A) \to \mathcal{U}$ with the following data:*

$$
\begin{aligned}
\mathsf{nil}^* &: P(\mathsf{nil}) \\
- ::^* - &: (a : A)\{as : \mathsf{cList}(A)\} \to P(as) \to P(a :: as) \\
\mathsf{comm}^* &: \{a\ b : A\}\{as\ bs\ cs : \mathsf{cList}(A)\} \\
&\quad \to \{pas : P(as)\}\{pbs : P(bs)\}\{pcs : P(cs)\} \\
&\quad \to (p : as = b :: cs) \to (pas =^P_p b ::^* pcs) \\
&\quad \to (q : a :: cs = bs) \to (a ::^* pcs =^P_q pbs) \\
&\quad \to a ::^* pas =^P_{\mathsf{comm}(p,q)} b ::^* pbs \\
\mathsf{trunc}^* &: (as : \mathsf{cList}(A)) \to \mathsf{isSet}(P(as))
\end{aligned}
$$

there is a unique function $f : (as : \mathsf{cList}(A)) \to P(as)$ with the following computation rules.

$$
\begin{aligned}
f(\mathsf{nil}) &\equiv nil* \\
f(a :: as) &\equiv a ::^* f(as) \\
\mathsf{apd}_f(\mathsf{comm}(p,q)) &= \mathsf{comm}^*(p, \mathsf{apd}_f(p), q, \mathsf{apd}_f(q))
\end{aligned}
$$