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 constructively 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 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) [47] embraces this proof-relevant notion of equality, and extends it with Voevodsky's univalence principle, 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/UF, 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. In this paper, we consider the problem of defining and studying *free commutative monoids* on sets in HoTT.

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 from S to the set of natural numbers $\mathbb{N}$. Alternatively, and informally, this is 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, but 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 various techniques for constructing well-behaved quotients (see Section 7 for a discussion).

Analogous to lists being free monoids, finite-multisets are free commutative monoids. We therefore define finite multisets in HoTT establishing the categorical 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 Ehrhard and Regnier's 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 commutative monad structure of the free commutative monoid on a set that 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 Girard's 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 Lafont 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 between concatenations and projections of multisets. Using this we establish the differential structure following Fiore's axiomatisation by constructing the creation operator.

- In Section 6, we characterise the path space of finite multisets using the commutation relation of creation/annihilation operators associated with the free commutative-monoid construction seen as a combinatorial Fock space. 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 (see Section 7.6). Informal paper proofs and proof sketches are also provided in the supplementary appendices.

1.3   Type theory

We choose to present our results in the type-theoretic language of the HoTT book [47]. All our claims will hold in a Cubical Type Theory [19,4,48] as well and our formalisation uses Cubical Agda [48]. 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 dependent product 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 $u : B(x)$, $v : B(y)$, we write $p_\star(u)$ for $\mathsf{transport}_B^p(u)$, and $u =_p^B v$ 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 $\mathsf{apd}_f : \prod_{x,y:A} (p : x =_A y) \to f(x) =_p^B f(y)$.

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 univalence/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 [47, 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 [47, 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 $M = (M; \cdot, e)$ is a set M with a commutative monoid structure, given by:*

 (i) *a multiplication function $- \cdot - : M \times M \to M$; and*

 (ii) *a unit element $e : M$;*

such that the following axioms hold:

 (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 2.2. *The natural numbers $\mathbb{N}$ [47, Section 1.9] are a commutative monoid under addition, with unit 0, and also under multiplication, with unit 1. For a set A, lists $\mathcal{L}(A)$ [47, Section 5.1] with the empty list for unit, and the append operation for multiplication are a monoid, but not generally a commutative monoid.*

Definition 2.3 (Homomorphism). *For commutative monoids $M = (M; \cdot_M, e_M)$ and $N = (N; \cdot_N, e_N)$, 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 $\mathsf{CMon}(M, N) :\equiv \sum_{f:M \to N} \mathsf{isCMonHom}(f)$.

Definition 2.4 (Equivalence). *Two commutative monoids $M = (M; \cdot_M, e_M)$ and $N = (N; \cdot_N, e_N)$, are equivalent as commutative monoids, $M \simeq_{\mathsf{CMon}} N$, whenever there is a commutative monoid homomorphism $f : M \to N$ which is an equivalence: $M \simeq_{\mathsf{CMon}} N :\equiv \sum_{f:M \to N} \mathsf{isCMonHom}(f) \wedge \mathsf{isEquiv}(f)$.*

Proposition 2.5 (Structure Identity Principle for commutative monoids). *Given two commutative monoids $M = (M; \cdot_M, e_M)$ and $N = (N; \cdot_N, e_N)$, the type of their equivalences is equivalent to their identity type: $(M \simeq_{\mathsf{CMon}} N) \simeq (M =_\mathcal{U} N)$.*

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

Definition 2.6 (Universal property of free commutative monoids). *For a set A, a commutative monoid $\mathcal{M}(A) = (\mathcal{M}(A); \oplus_A, \varnothing_A)$ 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_A 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 2.7. *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 2.8 (Free commutative-monoid monad). *For sets A and B,*

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

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

(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 [36].

Proposition 2.9. *For all sets A and B, $\tau_{A,B}{}^\sharp \circ \sigma_{A,\mathcal{M}(B)} = \sigma_{A,B}{}^\sharp \circ \tau_{\mathcal{M}(A),B} : \mathcal{M}(A) \times \mathcal{M}(B) \to \mathcal{M}(A \times B)$. This map is bilinear (that is, a homomorphism in each argument) and universal amongst all such.*

Example 2.10. *The free commutative monoid $\mathcal{M}(\mathbf{1})$ on $\mathbf{1}$ has the additive structure of natural numbers (Proposition 2.7). 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 2.9.

Proposition 2.11. *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*

$$\mathcal{M}(A) \times \mathcal{M}(B) \xrightarrow[\mathcal{M}(\imath_1) \times \mathcal{M}(\imath_2)]{\simeq} \mathcal{M}(A+B) \times \mathcal{M}(A+B) \xrightarrow{\oplus_{(A+B)}} \mathcal{M}(A+B) \quad , \quad \mathbf{1} \xrightarrow[\lambda(x:\mathbf{1}).\,\varnothing_\mathbf{0}]{\simeq} \mathcal{M}(\mathbf{0})$$

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 [47, 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 here 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 2.12 (ACM). *Let A be a type, we define the 1-HIT* $\mathsf{ACM}(A)$ *with the following point and path constructors:*

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

The first constructor η postulates that A maps to $\mathsf{ACM}(A)$. 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 $\mathsf{ACM}(A)$ to a set, since we wish to ignore any higher paths introduced by the path constructors. For completeness, we state the induction principle for ACM (Definition A.1).

$\mathsf{ACM}(A)$ 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 2.6).

Theorem 2.13. *For every set A, $\eta : A \to \mathsf{ACM}(A)$ 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 (also presented in [16]). Analogous to free monoids being given by finite lists (defined as an inductive type), free commutative monoids can be defined as lists, but identified up to swappings.

Definition 2.14 (sList). *The* swapped-list *over a type A is given by the type* $\mathsf{sList}(A)$*, which is the 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, this is a recursively defined HIT, where 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 in 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 2.15 (Propositional induction principle for sList)**.**
Let $P : \mathsf{sList}(A) \to \mathcal{U}$ *be a type family with the following data.*

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

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

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

Definition 2.16 (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 2.17 ($+\!\!\!+$)**.** *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*

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

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 2.15. Hence, the proofs of the monoid laws are simply the ones for lists.

Lemma 2.18. *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 2.19. *For* $x : A$ *and* $xs, ys : \mathsf{sList}(A)$, *we have* $x :: xs = xs +\!\!\!+ [x]$, *and* $xs +\!\!\!+ ys = ys +\!\!\!+ xs$.

Finally, one proves that sList satisfies the universal property of free commutative monoids (Definition 2.6).

Theorem 2.20. *For every set* A, $\eta : A \to \mathsf{sList}(A)$ *is the free commutative monoid on* A.

As both ACM and sList satisfy the same categorical universal property, it follows that they are equivalent as commutative monoids.

Corollary 2.21. *For every set A, we have an equivalence* $\mathsf{sList}(A) \simeq_{\mathsf{CMon}} \mathsf{ACM}(A)$.

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 ACM or sList using SIP (Proposition 2.5) and transport.

3 Generalized logic

This section, which stands in its own right, is an interlude to consider the category of relations in HoTT [47, Example 9.1.19]. 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 [40]. 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, one could turn it into the powerset monad using Voevodsky's *propositional resizing axiom* [49] to lower the universe level. Instead, we choose here to work with the power construction as a *relative monad* [2,28] and follow the framework of Kleisli bicategories [33,28] (restricted 1-categorically) 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* $\Bbbk_A : A \to \mathfrak{P}(A) : a \longmapsto \lambda(x : A).\, a =_A x$.

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

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

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

3.2 The category of relations

We define the category of relations [47, Example 9.1.19]) as the Kleisli category of $\mathfrak{P}$.

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

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

The category Rel has a symmetric *tensor* 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:

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

In fact, Rel is compact closed [22,35], 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

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

To summarise:

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

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

　　Note that $(-)_*$ preserves coproducts, which in Rel become biproducts. Therefore, as is well known, 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 $c :\equiv [\imath_2, \imath_1]$ 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 [32, Section 3]) developing it in HoTT from the perspective of generalized logic considered in the previous section. We establish that the compact closed category with finite biproducts Rel admits cofree commutative comonoids and is therefore a model of Girard's linear logic with linear exponential comonad structure [8,34] as first considered by Lafont [38].

　　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* [21].

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

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

Example 4.2. *Since the inclusion* $(-)_* : \mathsf{Set} \rightarrow \mathsf{Rel}$ *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,*

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

The universal property of the free commutative monoid lifts from Set to Rel.

Theorem 4.3. *For every set* A, *the set* $\mathcal{M}(A)$ *with multiplication* $m :\equiv (\oplus_A)_* : \mathcal{M}(A) \otimes \mathcal{M}(A) \rightarrowtail \mathcal{M}(A)$ *and unit* $e :\equiv (\lambda_{(x:\mathbf{1})}.\ \varnothing)_* : \mathbf{1} \rightarrowtail \mathcal{M}(A)$ *equipped with* $(\eta_A)_* : A \rightarrowtail \mathcal{M}(A)$ *is the free commutative monoid on* A *in* Rel.

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

Definition 4.4 (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):*

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

where $c :\equiv (\langle \pi_2, \pi_1 \rangle)_*$ is the symmetry isomorphism.

Example 4.5. *Each set has a unique product commutative comonoid structure in* Set*. This is preserved by the inclusion $(-)_* :$ Set $\to$ 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 4.6 (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 4.7. *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 4.8. *The compact closed category* 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

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

Proposition 4.9. *We have the following characterisation of the linear exponential monoidal structure:*

$$
\varphi = \big(\langle \mathcal{M}(\pi_1), \mathcal{M}(\pi_2) \rangle_*\big)^\dagger \,, \quad \phi = \big(\langle\,\rangle_*\big)^\dagger \,.
$$

The finite-multiset construction is canonically a strong symmetric monoidal endofunctor on Rel from the biproduct monoidal structure to the tensor monoidal structure (recall Propositions 2.11 and 3.6); the monoidal isomorphisms are the Seely (or storage) isomorphisms [46]:

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

5   Relational model of differential linear logic

We now give a formalisation of the relational model of Ehrhard and Regnier's differential linear logic (see, for instance, [25]). 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 2.10); namely, 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* [51], that is, nil is the only invertible element:

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

Note that the dependent product defining $\mathsf{isSing}(as)$ is not truncated and hence this type is not obviously a proposition. We can nevertheless show that there is a *unique choice* for such a witness, making it a proposition. For 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* $\eta : A \to \mathcal{M}(A)$ *is an embedding; that is,* $\mathsf{ap}_\eta : x = y \to [x] = [y]$ *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,

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

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 *and* B *be sets.*

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

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

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

5.2   Differential structure

Differential structure in categorical models of linear logic can be described in a variety of forms, see for instance [9,26,25]. Here, we follow the axiomatisation of Fiore [26] in the context of models with biadditive structure (that is, biproducts) as in Section 4.

In Rel, this differential structure consists of a natural transformation *creation map* [26, Definition 4.3]:

$$\eta_A : A \rightarrowtail \mathcal{M}(A)$$

subject to three laws as follows:

$$
\begin{array}{ccc}
& \mathcal{M}(A) & \\
\eta \nearrow & & \searrow \epsilon \\
A \xrightarrow[\text{id}]{} & & A
\end{array}
\qquad
\begin{array}{ccccc}
A & \xrightarrow{\eta} & \mathcal{M}(A) & \xrightarrow{\delta} & \mathcal{M}^2(A) \\
\simeq \downarrow & & & & \uparrow m \\
A \otimes \mathbf{1} & \xrightarrow[\eta \otimes e]{} & \mathcal{M}(A) \otimes \mathcal{M}(A) & \xrightarrow[\eta \otimes \delta]{} & \mathcal{M}^2(A) \otimes \mathcal{M}^2(A)
\end{array}
$$

$$
\begin{array}{ccc}
A \otimes \mathcal{M}(B) & \xrightarrow{\eta \otimes \text{id}} & \mathcal{M}(A) \otimes \mathcal{M}(B) \\
\text{id} \otimes \epsilon \downarrow & & \downarrow \varphi \\
A \otimes B & \xrightarrow{\eta} & \mathcal{M}(A \otimes B)
\end{array}
$$

This, as established in [26, Section 4 §*Differentiation*] (see also [9]) induces a *differential category* [10] 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 $a, a' : A$, we have $a = a' \iff [a] = [a']$.*

 (ii) *For $a : A$ and $s : \mathcal{M}^2(A)$, we have $[a] = \mu(s) \iff \exists_{(t:\mathcal{M}(\mathcal{M}(A)))}.\ \mu(t) = \mathsf{nil} \wedge [a] :: t = s$.*

(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. *The category* 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 6.8) 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* as developed by Fiore [26,27].

The following diagrams, involving the Seely isomorphisms (2), commute (up to homotopy)

$$
\begin{array}{ccc}
\mathcal{M}(A) \otimes \mathcal{M}(A) & \xrightarrow{\ \widetilde{\simeq}\ } & \mathcal{M}(A + A) \\
{}_{m}\searrow & \quad \swarrow {}_{\mathcal{M}(\nabla)} & \\
& \mathcal{M}(A) & \\
{}_{w}\swarrow & \quad \searrow {}_{\mathcal{M}(\Delta)} & \\
\mathcal{M}(A) \otimes \mathcal{M}(A) & \xleftarrow{\ \simeq\ } & \mathcal{M}(A + A)
\end{array}
\qquad
\begin{array}{ccc}
\mathbf{1} & \xrightarrow{\ \widetilde{\simeq}\ } & \mathcal{M}(\mathbf{0}) \\
{}_{e}\searrow & \quad \swarrow {}_{\mathcal{M}(u)} & \\
& \mathcal{M}(A) & \\
{}_{k}\swarrow & \quad \searrow {}_{\mathcal{M}(n)} & \\
\mathbf{1} & \xleftarrow{\ \simeq\ } & \mathcal{M}(\mathbf{0})
\end{array}
$$

and one can use the symmetric monoidal coherence laws to transport the biproduct commutative bialgebra structure through $\mathcal{M}$ as follows. We refer the reader to [26,27] for this theory.

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 4.3 and 4.8.*

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

$$
\begin{array}{ccccc}
\mathcal{M}(A) \otimes \mathcal{M}(A) & \xrightarrow{\;\;m\;\;} & \mathcal{M}(A) & \xrightarrow{\;\;w\;\;} & \mathcal{M}(A) \otimes \mathcal{M}(A) \\
{\scriptstyle w \otimes w}\downarrow & & & & \uparrow{\scriptstyle m \otimes m} \\
\mathcal{M}(A) \otimes \mathcal{M}(A) \otimes \mathcal{M}(A) \otimes \mathcal{M}(A) & \xrightarrow[\;\;\mathsf{id}_{\mathcal{M}(A)} \otimes c \otimes \mathsf{id}_{\mathcal{M}(A)}\;\;] & & & \mathcal{M}(A) \otimes \mathcal{M}(A) \otimes \mathcal{M}(A) \otimes \mathcal{M}(A)
\end{array}
$$

where c is the symmetry isomorphism. Spelling this out in elementary terms, we have that free commutative monoids satisfy the *Riesz refinement property* [23].

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

$$as \uplus bs = cs \uplus ds$$

$$\Longleftrightarrow$$

$$\exists_{(xs_1, xs_2, ys_1, ys_2 : \mathcal{M}(A))}. \; \big(as = xs_1 \uplus xs_2\big) \; \wedge \; \big(bs = ys_1 \uplus ys_2\big) \; \wedge \; \big(xs_1 \uplus ys_1 = cs\big) \wedge \big(xs_2 \uplus ys_2 = ds\big) \; .$$

Using this identity, we obtain the characterisation below of the equality type for non-empty finite multisets.

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

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

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:

$$
\frac{as = b :: cs \qquad a :: cs = bs}{a :: as = b :: bs} \; \mathsf{comm}
\tag{3}
$$

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

Example 6.4. *For $a, b, c : A$ and $cs, ds : \mathcal{M}(A)$,*

(i)

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

(ii)

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

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

Definition 6.5. *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:*

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

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

Example 6.6. *For* $a : A$*, the type* ($a :: a :: \text{nil} \sim_A a :: a :: \text{nil}$) *is inhabited by both of these terms:* $\text{cons-cong}(\text{refl}, \text{cons-cong}(\text{refl}, \text{nil-cong}))$ *and* $\text{comm-rule}(\text{cons-cong}(\text{refl}, \text{nil-cong}), \text{cons-cong}(\text{refl}, \text{nil-cong}))$.

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

Proposition 6.7. *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 6.8. *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 6.9. *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:*

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

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 the relation $\sim_A$ encodes permutations of the elements of the related lists; alternatively, permutations on the finite set of indices of the lists. We have an equivalence

$$\text{vec} : \mathcal{L}(A) \simeq \left(\textstyle\sum_{\ell:\mathbb{N}} \text{Fin}_\ell \to A \right) : \text{list}$$

and we now define a new relation $\approx_A$ on the above representation of lists by relating two listing functions if, and only if, there exists a permutation of their indices that shuffles one into the other.

Definition 6.10. *For a type A, define:*

$$- \approx_A - : \left(\textstyle\sum_{\ell:\mathbb{N}} \text{Fin}_\ell \to A \right) \to \left(\textstyle\sum_{\ell:\mathbb{N}} \text{Fin}_\ell \to A \right) \to \mathcal{U} \ ,$$
$$(m, f) \approx_A (n, g) :\equiv (\phi : \text{Fin}_m \xrightarrow{\sim} \text{Fin}_n) \times (f = g \circ \phi) \ .$$

Since permutations are simply invertible functions, it is straightforward to establish that this is an equivalence relation. We now show that the relation $\approx_A$ *extensionally* agrees with our relation $\sim_A$. 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 6.11. *For $as, bs : \mathcal{L}(A)$, we have*

$$\mathsf{eval} : \; as \sim_A bs \to \mathsf{vec}(as) \approx_A \mathsf{vec}(bs)$$

and, for $(m, f), (n, g) : \left(\sum_{\ell:\mathbb{N}} \mathsf{Fin}_\ell \to A \right)$, we have

$$\mathsf{quote} : \; (m, f) \approx_A (n, g) \to \mathsf{list}(m, f) \sim_A \mathsf{list}(n, g) \;\; .$$

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 6.12. *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 valued in propositions, since there may be multiple inhabitants that encode the same permutation of the underlying elements (see Example 6.6). We propositionally truncate it, defining $xs \; \bar{\sim}_A \; ys :\equiv \|xs \sim_A ys\|_{-1}$, before using it in as a quotient. The resulting effective quotient of lists satisfies the categorical universal property of free commutative monoids.

Theorem 6.13. *For a set A, the composite $A \to \mathcal{L}(A) \to \mathcal{L}(A)_{/\bar{\sim}_A}$ is the free commutative monoid on A.*

Corollary 6.14. *For every set A, we have the equivalence $\mathcal{L}(A)_{/\bar{\sim}_A} \simeq_{\mathsf{CMon}} \mathsf{ACM}(A)$.*

6.3   Commuted-list construction

As a final application of the commutation relation (Theorem 6.3), we give a construction of another HIT, cList (Definition 6.15), for finite multisets that uses the comm-rule (3) as a path constructor. This is an example of a conditional path constructor, which corresponds to a quasi-equational (or Horn) theory.

Definition 6.15 (cList). *For a type A, the 1-HIT $\mathsf{cList}(A)$ is given by the following point and path constructors:*

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

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

Proposition 6.16. *For every set A, $\mathsf{cList}(A)$ is a commutative monoid.*

Theorem 6.17. *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. A preliminary version of our work was presented in [16].

7.1   Finite multisets

The simplest encoding of finite multisets is lists up to permutation of their elements. In this context, the relation of Definition 6.10 using automorphisms on finite cardinals as permutations has been considered by Altenkirch et al. [1] and Nuo [41]; while Danielsson [20] has considered a similar relation for bag equivalence of lists. Multisets in type theory have also been studied by Gylterud [31] 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. Using HITs in HoTT/UF, Angiuli et al. [5] considered 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; these are shown equivalent. They are interested in the representation independence of data structures and assume decidable equality on the carrier set.

7.2   Commutation axioms

The definition of various (higher) algebraic structures in HoTT/UF using HITs is an active line of research; for example, the problem of defining (free higher) groups has been considered by [37,14,3]. For algebraic structures with commutation axioms, there are various techniques which are known folklore.

Basold et al. [6] and Frumin et al. [30] consider Kuratowski-finite sets in HoTT defined as free join semilattices using the universal-algebraic construction. Our universal-algebraic construction of free commutative monoids (Definition 2.12) is similar to their HIT $\mathcal{K}(A)$ but lacking the idempotence axiom. The swapped-list construction (Definition 2.14, or listed finite multisets) is similar to their HIT $\mathcal{L}(A)$ (or listed finite sets) but lacking the duplication axiom. Frumin et al. [30, Theorem 2.8] show that these two HITs are equivalent by constructing an explicit equivalence. In contrast, we prove the categorical universal property of free commutative monoids for each HIT, thereby getting their equivalence for free.

Formalisations of free abelian monoids and free abelian groups in HoTT/UF also appear in the UniMath [50], HoTT [7], and HoTT-Agda [11] libraries; they use various forms of commutation axioms. In UniMath, free abelian monoids are defined by quotienting free monoids (lists) by the relation $g \sim h :\equiv \exists_{(x,y)}.\, g = x \mathbin{+\!\!+} y \wedge y \mathbin{+\!\!+} x = h$. In the HoTT (Coq) library, abelianisation is defined using a homotopy coequaliser (or, equivalently, a HIT) where the commutation axiom is essentially defined as $\forall_{(x,y,z)}.\, x \cdot (y \cdot z) = x \cdot (z \cdot y)$. In HoTT-Agda, free abelian groups are defined by abelianising the free group, essentially using the commutation relation $l_1 \mathbin{+\!\!+} l_2 = l_2 \mathbin{+\!\!+} l_1$ on lists. We use the obvious commutation axiom in the universal-algebraic construction, and adjacent transpositions (swap) in the swapped-list construction, which is enough to prove the categorical universal property and the structural properties that we use in our applications. The commutation relation and the conical refinement monoid relations that we prove in Section 6.1 have not been considered before.

7.3   Higher Inductive Types

All three HITs, ACM (Definition 2.12), sList (Definition 2.14), and cList (Definition 6.15) are recursive HITs, since the point and path constructors refer to points in the type. However, cList is a recursive HIT with a conditional path constructor comm, since it refers to the path space of the type. These definitions of HITs are accepted by (the current version of) Cubical Agda [48]. However, it is unclear whether the known schemas for HITs in cubical type theories can encode conditional path constructors. Fiore et al. [29] consider sList in their quotient-inductive types framework and mention cList as an example that goes beyond it.

7.4   Relational model of linear logic

Standard presentations of the relational model of linear logic, see e.g. [12, Section 4], [24, Section 2.1], [39, Section III.B], or [43, Section II.A], are typically written in informal mathematical vernacular and often implicitly assume further structure on sets when manipulating the finite-multiset construction. For instance, finite multisets may be listed, thereby implicitly relying on an underlying order, and the permutation invariance of constructions may be glossed over; or they may be represented as finitely supported

$\mathbb{N}$-valued functions sometimes relying on the decidability of equality and without dwelling on notions of finiteness. On the other hand, our constructive treatment of the subject has forced us to be completely rigorous in all respects.

7.5   Groupoidification

This work is about 0-truncated finite multisets, which only allow elimination into sets. This is somewhat restrictive. The preliminary presentation of our work [16] 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 hexagon path constructor to establish the coherence of swap.

It is folklore that 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 $\mathcal{U}_{\mathsf{Fin}} :\equiv \sum_{X:\mathcal{U}} \sum_{n:\mathbb{N}} \|X = \mathsf{Fin}_n\|_{-1}$ (see, for instance, [13]). 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 [44,45]. 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. [18,17] in the context of building fully-abstract models for reversible programming languages [18,15]. We believe that it is possible to apply these techniques to generalise the work presented in this paper to groupoids.

7.6   Formalisation

As mentioned in the introduction, we have a partial formalisation of the results in this paper using Cubical Agda. The formalisation is around 3000 lines of code and is available at https://github.com/vikraman/generalised-species/tree/master/set. The status of the formalisation is summarised in Table F.1 in the appendix. Some key highlights and differences with the paper are:

- In the definition of commutative monoid structure, we drop one of the unit axioms since it can be derived from commutativity.

- The Seely isomorphism and commutative monad structure can be proved using the categorical universal property; however, we sometimes work with the type-theoretic induction principle of sList (Definition A.2 and Lemma 2.15) because it is easier to formalise.

- When working with lists and permutations, we use different, more convenient representations of $\mathcal{L}$ and Fin, these are equivalent and the results can be transported.

- The HIT for the swapped-list construction was merged into the Cubical Agda library [42], and various other results about finite multisets and free commutative monoids have been formalised by contributors to the library.

References

[1] Altenkirch, T., T. Anberrée and N. Li, *Definable Quotients in Type Theory*.
URL http://www.cs.nott.ac.uk/~psztxa/publ/defquotients.pdf

[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 (2010), 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 (2020), pp. 67–73.

[4] Angiuli, C., "Computational Semantics of CartesianCubical Type Theory," Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA (2019).
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*, Proceedings of the ACM on Programming Languages **5** (2021), pp. 12:1–12:30.

[6] Basold, H., H. Geuvers and Niels van der Weide, *Higher Inductive Types in Programming* (2017).
URL http://www.jucs.org/doi?doi=10.3217/jucs-023-01-0063

[7] Bauer, A., J. Gross, P. L. Lumsdaine, M. Shulman, M. Sozeau and B. Spitters, *The HoTT library: A formalization of homotopy type theory in Coq*, in: *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017 (2017), pp. 164–172.

[8] Benton, N., G. Bierman, V. de Paiva and J. M. E. 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 (1993), pp. 61–84.

[9] Blute, R. F., J. R. B. Cockett, J.-S. P. Lemay and R. A. G. Seely, *Differential Categories Revisited*, Applied Categorical Structures **28** (2020), pp. 171–235.

[10] Blute, R. F., J. R. B. Cockett and R. A. G. Seely, *Differential categories*, Mathematical Structures in Computer Science **16** (2006), pp. 1049–1083.
URL https://www.cambridge.org/core/journals/mathematical-structures-in-computer-science/article/differential-categories/794F222FE0710CBEF06492127AF4E5CF

[11] Brunerie, G., 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*, available at https://github.com/HoTT/HoTT-Agda.
URL https://github.com/HoTT/HoTT-Agda

[12] Bucciarelli, A., T. Ehrhard and G. Manzonetto, *Not Enough Points Is Enough*, in: J. Duparc and T. A. Henzinger, editors, *Computer Science Logic*, LNCS **4646**, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007 pp. 298–312.

[13] Buchholtz, U., *Genuine pairs and the trouble with triples in homotopy type theory* (2021).
URL https://types21.liacs.nl/timetable/event/genuine-pairs-and-the-trouble-with-triples-in-homotopy-type-theory/

[14] 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 (2018), pp. 205–214.

[15] Carette, J., C.-H. Chen, V. Choudhury and A. Sabry, *From Reversible Programs to Univalent Universes and Back*, Electronic Notes in Theoretical Computer Science **336** (2018), pp. 5–25.
URL https://linkinghub.elsevier.com/retrieve/pii/S1571066118300161

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

[17] Choudhury, V. and J. Karwowski, *Artifact for Symmetries in Reversible Programming*, Zenodo (2021).

[18] Choudhury, V., J. Karwowski and A. Sabry, *Symmetries in reversible programming: From symmetric rig groupoids to reversible programming languages*, Proceedings of the ACM on Programming Languages **6** (2022), pp. 6:1–6:32.

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

[20] 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 (2012), pp. 149–165.

[21] Day, B. J., *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 (1970), pp. 1–38.

[22] Day, B. J., *Note on compact closed categories*, Journal of the Australian Mathematical Society **24** (1977), pp. 309–311.
URL https://www.cambridge.org/core/journals/journal-of-the-australian-mathematical-society/article/note-on-compact-closed-categories/F36D383A92F41665ED4BBE9F8F199B20

[23] Dobbertin, H., *Refinement monoids, Vaught monoids, and Boolean algebras*, Mathematische Annalen **265** (1983), pp. 473–487.

[24] Ehrhard, T., *The Scott model of linear logic is the extensional collapse of its relational model*, Theoretical Computer Science **424** (2012), pp. 20–45.
URL https://www.sciencedirect.com/science/article/pii/S0304397511009467

[25] Ehrhard, T., *An introduction to differential linear logic: Proof-nets, models and antiderivatives*, Mathematical Structures in Computer Science **28** (2018), 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

[26] 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 (2007), pp. 163–177.

[27] Fiore, M. P., *An Axiomatics and a Combinatorial Model of Creation/Annihilation Operators*, arXiv:1506.06402 [math] (2015).
URL http://arxiv.org/abs/1506.06402

[28] Fiore, M. P., N. Gambino, J. M. E. Hyland and G. Winskel, *Relative pseudomonads, Kleisli bicategories, and substitution monoidal structures*, Selecta Mathematica **24** (2018), pp. 2791–2830.

[29] Fiore, M. P., A. M. Pitts and S. C. Steenkamp, *Quotients, inductive types, and quotient inductive types*, Logical Methods in Computer Science **Volume 18, Issue 2** (2022), p. 7076.
URL https://lmcs.episciences.org/7076

[30] 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 (2018), pp. 201–214.

[31] Gylterud, H. R., *Multisets in type theory*, Mathematical Proceedings of the Cambridge Philosophical Society **169** (2020), pp. 1–18.
URL https://www.cambridge.org/core/journals/mathematical-proceedings-of-the-cambridge-philosophical-society/article/abs/multisets-in-type-theory/C17604B9927E477B70126529A2A91321

[32] Hyland, J. M. E., *Some reasons for generalising domain theory*, Mathematical Structures in Computer Science **20** (2010), pp. 239–265.
URL https://www.cambridge.org/core/product/identifier/S0960129509990375/type/journal_article

[33] Hyland, J. M. E., *Elements of a theory of algebraic theories*, Theoretical Computer Science **546** (2014), pp. 132–144.

[34] Hyland, J. M. E. and A. Schalk, *Glueing and orthogonality for models of linear logic*, Theoretical Computer Science **294** (2003), pp. 183–231.
URL https://www.sciencedirect.com/science/article/pii/S0304397501002419

[35] Kelly, G. M. and M. L. Laplaza, *Coherence for compact closed categories*, Journal of Pure and Applied Algebra **19** (1980), pp. 193–213.
URL https://www.sciencedirect.com/science/article/pii/0022404980901012

[36] Kock, A., *Monads on symmetric monoidal closed categories*, Archiv der Mathematik **21** (1970), pp. 1–10.

[37] 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 (2018), pp. 599–608.

[38] Lafont, Y., *The linear abstract machine*, Theoretical Computer Science **59** (1988), pp. 157–180.
URL https://www.sciencedirect.com/science/article/pii/0304397588901004

[39] Laird, J., G. Manzonetto, G. McCusker and M. Pagani, *Weighted Relational Models of Typed Lambda-Calculi*, in: *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2013, pp. 301–310.

[40] Lawvere, F. W., *Metric spaces, generalized logic, and closed categories*, Rendiconti del Seminario Matematico e Fisico di Milano **43** (1973), pp. 135–166.

[41] Li, N., *Quotient types in type theory* (2015).
URL http://eprints.nottingham.ac.uk/28941/

[42] Mörtberg, A., A. Vezzosi, E. Cavallo et al., *A standard library for Cubical Agda*, Agda Github Community.
URL https://github.com/agda/cubical

[43] Ong, C.-H. L., *Quantitative semantics of the lambda calculus: Some generalisations of the relational model*, in: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2017, pp. 1–12.

[44] Piceghello, S., *Coherence for symmetric monoidal groupoids in HoTT/UF* (2019), p. 2.
URL http://www.ii.uib.no/~bezem/abstracts/TYPES_2019_paper_10

[45] Piceghello, S., "Coherence for Monoidal and Symmetric Monoidal Groupoids in Homotopy Type Theory," The University of Bergen, 2021.
URL https://bora.uib.no/bora-xmlui/handle/11250/2830640

[46] Seely, R. A. G., *Linear Logic, *-Autonomous Categories and Cofree Coalgebras*, in: *In Categories in Computer Science and Logic* (1989), pp. 371–382.

[47] Univalent Foundations Program, T., "Homotopy Type Theory: Univalent Foundations of Mathematics," Univalent Foundations Program, Institute for Advanced Study, 2013.
URL https://homotopytypetheory.org/book

[48] Vezzosi, A., A. Mörtberg and A. Abel, *Cubical agda: A dependently typed programming language with univalence and higher inductive types*, Proceedings of the ACM on Programming Languages **3** (2019), pp. 87:1–87:29.

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

[50] Voevodsky, V., B. Ahrens, D. Grayson et al., *UniMath — a computer-checked library of univalent mathematics*, available at https://unimath.org.
URL https://github.com/UniMath/UniMath

[51] Wehrung, F., *Tensor products of structures with interpolation*, Pacific Journal of Mathematics **176** (1996), pp. 267–285.
URL http://msp.org/pjm/1996/176-1/p16.xhtml

A   Supplementary material for Section 2

Proposition 2.11. *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*

$$\mathcal{M}(A) \times \mathcal{M}(B) \xrightarrow[\mathcal{M}(\imath_1) \times \mathcal{M}(\imath_2)]{\simeq} \mathcal{M}(A+B) \times \mathcal{M}(A+B) \xrightarrow{\oplus_{(A+B)}} \mathcal{M}(A+B) \quad , \quad \mathbf{1} \xrightarrow[\lambda(x:\mathbf{1}).\, \varnothing_{\mathbf{0}}]{\simeq} \mathcal{M}(\mathbf{0})$$

Proof. Being the product of two commutative monoids, $\mathcal{M}(A) \times \mathcal{M}(B)$ is a commutative monoid and one obtains the inverse $\mathcal{M}(A+B) \to \mathcal{M}(A) \times \mathcal{M}(B)$ of the given map $\mathcal{M}(A) \times \mathcal{M}(B) \to \mathcal{M}(A+B)$ by homomorphically extending the map

$$\left[\; \lambda(a:A).\,(\,\eta(a), \varnothing\,) \;\; | \;\; \lambda(b:B).\,(\,\varnothing, \eta(b)\,) \; \right] : A + B \to \mathcal{M}(A) \times \mathcal{M}(B) \; .$$

On the other hand, the unique map $\mathcal{M}(\mathbf{0}) \to \mathbf{1}$ is the inverse of the given map $\mathbf{1} \to \mathcal{M}(\mathbf{0})$. $\qquad\square$

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

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

there is a unique function $f : (x : \mathsf{ACM}(A)) \to P(x)$ *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 2.13. *For every set* A, $\eta : A \to \mathsf{ACM}(A)$ *is the free commutative monoid on* A.

Proof. Given $f : A \to M$, we define $f^\sharp : \mathsf{ACM}(A) \to M$ 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. $\qquad\square$

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

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

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

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

Lemma 2.15 (Propositional induction principle for sList)**.**
Let $P : \mathsf{sList}(A) \to \mathcal{U}$ *be a type family with the following data.*

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

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

Proof. Using the standard induction principle for sList, we need 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 2.18. *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 2.19. *For* $x : A$ *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

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

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

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

□

Proposition A.3. *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 let

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

On the path constructor, we have

$$f(x) \cdot (f(y) \cdot f^\sharp(xs))$$
$$= (f(x) \cdot f(y)) \cdot f^\sharp(xs) \qquad\qquad \text{associativity}$$
$$= (f(y) \cdot f(x)) \cdot f^\sharp(xs) \qquad\qquad \text{commutativity}$$
$$= f(y) \cdot (f(x) \cdot f^\sharp(xs)) \qquad\qquad \text{associativity}$$

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

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

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

□

Theorem 2.20. *For every set A, $\eta : A \to \mathsf{sList}(A)$ is the free commutative monoid on A.*

Proof. We get the $(-)^\sharp$ operation by Proposition A.3. We need to show that this 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 2.21. *For every set A, we have an equivalence* $\mathsf{sList}(A) \simeq_{\mathsf{CMon}} \mathsf{ACM}(A)$.

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

B Supplementary material for Section 3

Proposition 3.3. *The following identities hold.*

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

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

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

Proof.

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

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

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

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

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

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

$\square$

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

Proof. We note that $\mathsf{Hom}_{\mathsf{Rel}}(A, B) :\equiv A \rightarrowtail B$ is an hSet since hProp is. The identity arrow in $\mathsf{Hom}(A, A)$ is given by $\natural_A$. The composition of $g : B \rightarrowtail C$ and $f : A \rightarrowtail B$ is given by $g^* \circ f$. The unit and associativity laws follow from Proposition 3.3. To show that Rel is univalent, see [47, Example 9.1.19]. $\square$

C Supplementary material for Section 4

Theorem 4.3. *For every set A, the set $\mathcal{M}(A)$ with multiplication $m :\equiv (\oplus_A)_* : \mathcal{M}(A) \otimes \mathcal{M}(A) \rightarrowtail \mathcal{M}(A)$ and unit $e :\equiv (\lambda_{(x:\mathbf{1})}.\ \varnothing)_* : \mathbf{1} \rightarrowtail \mathcal{M}(A)$ equipped with $(\eta_A)_* : A \rightarrowtail \mathcal{M}(A)$ is the free commutative monoid on A in* Rel.

Proof. For a commutative monoid M in Rel and $f : A \twoheadrightarrow M$, one needs to define a unique homomorphic extension $\mathcal{M}(A) \twoheadrightarrow M$. 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. $\square$

Corollary 4.7. *For every set A, $\epsilon_A :\equiv ((\eta_A)_*)^\dagger : \mathcal{M}(A) \twoheadrightarrow A$ is the cofree commutative comonoid on A in* Rel; *that is, for a commutative comonoid K, every relation $r : K \twoheadrightarrow A$ has a unique homomorphic coextension $r_\sharp : K \twoheadrightarrow \mathcal{M}(A)$ over ϵ_A.*

Proof. This follows from Theorem 4.3 by self duality of Rel. For a commutative comonoid (K, w, k) in Rel and $r : K \twoheadrightarrow A$, the homomorphic coextension $r_\sharp : K \twoheadrightarrow \mathcal{M}(A)$ is $((r^\dagger)^\sharp)^\dagger$ where the homomorphic extension is taken with respect to the promonoidal convolution on the commutative monoid $(K; w^\dagger, e^\dagger)$. $\square$

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 $\eta : A \to \mathcal{M}(A)$ is an embedding; that is, $\mathsf{ap}_\eta : x = y \to [x] = [y]$ 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) 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.

$\square$

Proposition 5.5. *For a set A,*

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

Proof. Follows from Proposition 5.4 using the unique choice function. $\square$

Proposition 5.6. *Let A and B be sets.*

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

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

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

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. $\square$

E   Supplementary material for Section 6

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

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

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

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

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

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

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

$$(a :: as) = (bs \mathbin{+\!\!+} cs)$$

$$\iff$$

$$\exists_{(xs:\mathcal{M}(A))}.(a :: xs = bs) \land (as = xs \mathbin{+\!\!+} cs) \lor \exists_{(ys:\mathcal{M}(A))}.(as = bs \mathbin{+\!\!+} ys) \land (a :: ys = cs) \ .$$

$\square$

Definition 6.11. *For $as, bs : \mathcal{L}(A)$, we have*

$$\mathsf{eval} : \ as \sim_A bs \to \mathsf{vec}(as) \approx_A \mathsf{vec}(bs)$$

and, for $(m, f), (n, g) : \left(\sum_{\ell:\mathbb{N}} \mathsf{Fin}_\ell \to A \right)$, we have

$$\mathsf{quote} : \ (m, f) \approx_A (n, g) \to \mathsf{list}(m, f) \sim_A \mathsf{list}(n, g) \ .$$

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

To construct quote, we start with $(m, f) \approx_A (n, g)$; that is, we have a $\phi : \mathsf{Fin}_m \simeq \mathsf{Fin}_n$ such that $f = g \circ \phi$. Knowing ϕ, we deduce that $m = n$ and proceed by induction on $m : \mathbb{N}$.

Case 0: Follows by $\mathsf{nil\text{-}cong}$.

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

Case $\mathsf{S}(\mathsf{S}(\ell))$: We use the decidable equality on $\mathsf{Fin}_{\mathsf{S}(\mathsf{S}(\ell))}$ to test whether or not $\phi(0) = 0$.

Case $\phi(0) = 0$: We get a path between the elements $f(0) = g(\phi(0)) = g(0)$. We also have $(\mathsf{S}(\ell), f \circ \mathsf{S}) \approx_A (\mathsf{S}(\ell), g \circ \mathsf{S})$ and recursively obtain $\mathsf{list}(\mathsf{S}(\ell), f \circ \mathsf{S}) \sim_A \mathsf{list}(\mathsf{S}(\ell), g \circ \mathsf{S})$. Finally, we combine them using $\mathsf{cons\text{-}cong}$.

Case $\phi(0) = \mathsf{S}(k)$: We need the following auxiliary definition: For $t : \mathsf{Fin}_{\mathsf{S}(i)}$, we let $\widehat{t} : \mathsf{Fin}_i \to \mathsf{Fin}_{\mathsf{S}(i)}$ be given by $\widehat{t}(j) = j$ for $j < t$ and by $\widehat{t}(j) = \mathsf{S}(j)$ for $j \geq t$. We then have that

$$(\mathsf{S}(\ell), f \circ \mathsf{S}) \approx_A (\mathsf{S}(\ell), g \circ \widehat{\phi(0)})$$

and therefore obtain

$$
\begin{aligned}
&\mathsf{list}(\mathsf{S}(\ell), f \circ \mathsf{S}) \\
\sim_A \ &\mathsf{list}(\mathsf{S}(\ell), g \circ \widehat{\phi(0)}) && \text{, by recursion} \\
= \ &g(0) :: \mathsf{list}(\ell, g \circ \widehat{\mathsf{S}(k)} \circ \mathsf{S}) \\
= \ &g(0) :: \mathsf{list}(\ell, g \circ \mathsf{S} \circ \widehat{k})
\end{aligned}
$$

(E.1)

On the other hand,

$$\left(\, \mathsf{S}(\ell)\,,\, [\,0 \mapsto g(\phi(0)) \mid \mathsf{S}(i) \mapsto g(\mathsf{S}(\widehat{k}(i)))\,]\,\right) \;\approx_A\; \left(\, \mathsf{S}(\ell)\,,\, g \circ \mathsf{S}\,\right)$$

and we further obtain

$$
\begin{aligned}
&\quad f(0) \;::\; \mathsf{list}(\,\ell\,,\, g \circ \mathsf{S} \circ \widehat{k}\,) \\
&= \;\; g(\phi(0)) \;::\; \mathsf{list}(\,\ell\,,\, g \circ \mathsf{S} \circ\widehat{k}\,) \\
&= \;\; \mathsf{list}(\,\mathsf{S}(\ell)\,,\, [\,0 \mapsto g(\phi(0)) \mid \mathsf{S}(i) \mapsto g(\mathsf{S}(\widehat{k}(i)))\,]\,) \\
&\sim_A \; \mathsf{list}(\,\mathsf{S}\,(\ell)\,,\, g \circ \mathsf{S}\,) \qquad\qquad\qquad\qquad\qquad\text{, by recursion}
\end{aligned}
\tag{E.2}
$$

Finally, an application of the comm-rule to (*Eq.* (E.1)) and (*Eq.* (E.2)) yields

$$\mathsf{list}(\,\mathsf{S}(\mathsf{S}\,(\ell))\,,\, f\,) \;=\; f(0) \;::\; \mathsf{list}(\,\mathsf{S}(\ell)\,,\, f \circ \mathsf{S}\,) \;\sim_A\; g(0) \;::\; \mathsf{list}(\,\mathsf{S}(\ell)\,,\, g \circ \mathsf{S}\,) \;=\; \mathsf{list}(\,\mathsf{S}(\mathsf{S}(\ell))\,,\, g\,)$$

as required.

$\square$

Lemma 6.12. *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 6.11 and using the transitivity of the relation $\approx_A$. The rest is straightforward to establish. $\square$

Definition E.1 (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.

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

F   Supplementary material for Section 7

At the time of writing, the following results have been fully formalised, while the remaining ones have only been partially formalised.

`set.CMon`	universal-algebraic construction (Section 2.1)
`set.MSet`	swapped-list construction (Section 2.2)
`set.NSet`	commuted-list construction (Section 6.3)
`set.QSet`	quotiented-list construction (Section 6.3)
`SIP`	SIP for commutative monoids (Proposition 2.5)
`MSet.Length`, `MSet.Paths`	subsingleton multisets (Section 5.1)
`MSet.Cover`	commutation relation and path space (Section 6.1)
`Monoidal`	Seely isomorphism (Proposition 2.11)
`Comm`	commutative monad structure (Proposition 2.9)
`Power`, `nPSh`, `PSh`	power objects and presheaves (Section 3.1)
`hRel`	category of relations (Section 3.2)
`Diff`	differential structure (Section 5.2)

Table F.1
Status of formalised results