Degrading Lists

Dylan McDermott
Reykjavík University
Reykjavík, Iceland
dylanm@ru.is

Maciej Piróg
Wrocław University
Wrocław, Poland
mpirog@cs.uni.wroc.pl

Tarmo Uustalu
Reykjavík University
Reykjavík, Iceland
Tallinn University of Technology
Tallinn, Estonia
tarmo@ru.is

ABSTRACT

We discuss the relationship between monads and their known generalisation, graded monads, which are especially useful for modelling computational effects equipped with a form of sequential composition. Specifically, we ask if a graded monad can be extended to a monad, and when such a degrading is in some sense canonical. Our particular examples are the graded monads of lists and non-empty lists indexed by their lengths, which gives us a pretext to study the space of all (non-graded) monad structures on the list and non-empty list endofunctors. We show that, in both cases, there exist infinitely many monad structures. However, while there are at least two ways to complete the graded monad structure on length-indexed lists to a monad structure on the list endofunctor, such a completion for non-empty lists is unique.

CCS CONCEPTS

• **Theory of computation → Functional constructs**; **Program semantics**.

KEYWORDS

monads, algebraic theories, graded monads, degrading, lists

ACM Reference Format:
Dylan McDermott, Maciej Piróg, and Tarmo Uustalu. 2020. Degrading Lists. In *22nd International Symposium on Principles and Practice of Declarative Programming (PPDP '20), September 8–10, 2020, Bologna, Italy.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3414080.3414084

1 INTRODUCTION

The term 'list monad' (on the category of sets and functions) usually describes the monad that arises from free monoids, that is, the one used in programming to express nondeterministic computations alongside the finite multiset and powerset monads. However, as we prove in this paper, there are actually infinitely many ways to turn the list endofunctor into a monad. Similarly, the usual 'non-empty list monad' is just one of infinitely many monads one can define on the endofunctor of non-empty lists. Thus, in broader contexts, one could speak in the plural about list *monads* and non-empty list *monads*. The goal of this paper is to study these structures from a more combinatorial angle. The results we obtain are quite surprising: the abundance and diversity of monads in both cases have defied our original intuitions.

Our more practical motivation is to use these results to investigate properties of a known generalisation of monads, *graded monads*. Introduced by Smirnov [23], they are useful for quantitative modelling of effects, e.g., with graded lists we can precisely fix or upper-bound the number of outcomes from a nondeterministic computation. Some examples include the semantics of type-and-effect systems for program analysis and transformation [8, 18] (where 'effect' refers to what is called 'grade' here), process semantics [6, 17], and tensorial logic [15]. Broadly speaking, a graded monad T is a family of endofunctors T_g, where the indices g (grades) are elements of a preordered monoid $\mathcal{G}$, and the unit (return) and multiplication (join) are appropriately coherent with the structure of $\mathcal{G}$; see Section 2 for a formal definition.

While every monad is trivially graded by a one-element monoid, in this paper we discuss the relationship in the opposite direction: how to turn a graded monad into a (non-graded) monad, and, if there are multiple such *degradings*, whether there exists one that is—in an appropriate sense—canonical. This question was asked by Fritz and Perrone [7], who hinted at *algebraic Kan extensions* [11, 16, 25] as the correct notion of degrading, and used the graded monad of length-indexed lists as the motivating example, but did not claim or prove anything about it.

We draw inspiration from dependently-typed programming, where it is a common pattern to turn an indexed type into a regular one (for example, "vectors", i.e., length-indexed lists, into lists) simply by hiding the index under an existential quantifier. The category-theoretic analogue of this construction is the appropriate colimit. As our main running example, we consider the graded monad $\mathrm{List}_=$ of length-indexed lists, with $\mathcal{G}$ the monoid of natural numbers with multiplication. We define the multiplication of a list of lists in the familiar fashion, as the concatenation of all inner lists. The colimit of $n \mapsto \mathrm{List}_{=n}$ is a functor that takes a set X to the disjoint union of $\mathrm{List}_{=n}X$ for all $n \in \mathbb{N}$, that is, the list functor List. Similarly, for the graded monad $\mathrm{List}_{+=}$ of length-indexed non-empty lists, the colimit is the non-empty list functor List_+.

The colimit construction works well on functors, but we also need to account for monad structure. For the colimit $\hat{T}$ of $g \mapsto T_g$ to be an algebraic Kan extension, the functor $\hat{T}$ has to carry a monad structure compatible with the graded monad structure of T, and the unique natural transformation from $\hat{T}$ into any other degrading should preserve the monad structure. However, it turns out that neither List nor List_+ enjoy these properties. We therefore look for something weaker.

PPDP '20, September 8–10, 2020, Bologna, Italy

ACM ISBN 978-1-4503-8821-4/20/09…$15.00
https://doi.org/10.1145/3414080.3414084

As our first candidate, we propose a notion of *shallow degrading*, which we use for any monad structure on the colimit that is compatible with the original graded monad structure. Then, we ask if some shallow degrading is unique. Uniqueness makes it canonical in the banal sense of there being no other option. This retains the existential quantifier intuition, the difference is that the unique maps are not required to preserve the monad structure. For lists and non-empty lists, our combinatorial results come in handy here. We show that there are at least two monad structures on List that agree with the graded monad $\mathrm{List}_{=}$, and hence there is no unique shallow degrading of possibly-empty lists. The situation is different for non-empty lists. As the main technical result of this paper, we show that exactly one monad structure on List_{+} (the usual one) is compatible with the graded monad $\mathrm{List}_{+=}$, and so the usual non-empty list monad is a unique shallow degrading. The proof is quite involved, and requires steps like a thorough computer search for possible 'prefixes' of the monad multiplication for lists of length 6. While we of course cannot know if a simpler proof of this fact is possible, we draw a conclusion from this situation that we should not hope for a simple generic condition telling us when a given graded monad can be uniquely degraded using this construction.

A second way of weakening the notion of algebraic Kan extension is to drop the requirement that we use the colimit of T. This leads to a notion of *initial degrading*: a degrading such that there is a unique structure-preserving map into any other degrading. We show that there is an initial degrading of the graded non-empty list monad. However, this degrading is not a non-empty list monad.

Contributions. To sum up, we make the following contributions:

- In Section 2, we introduce the notion of *degrading* of a graded monad. We discuss the colimit construction and the problem of furnishing it with a (unique) monad structure.
- In Section 3, we prove that there are infinitely many monad structures on the list endofunctor. Among them, there are two compatible with the multiplication of the graded list monad. From that, we conclude that $\mathrm{List}_{=}$ has no unique shallow degrading.
- In Section 4, we prove that, in the case of List_{+}, there are also infinitely monad structures, and that there are some that are not simply projections of monads on List.
- In Section 5, we prove that, despite the abundance of monad structures on List_{+}, only one is compatible with the usual one for balanced lists, yielding a unique shallow degrading.
- Finally, in Section 6, we introduce the notion of *initial* degrading. We construct such a degrading for the graded non-empty list monad and show that it is different from the shallow one.

While we use some category-theoretic concepts, we focus on set-theoretic lists. We have made efforts to make this paper accessible to a reader with only a basic understanding of category theory.

2 DEGRADING GRADED MONADS

In this section, we give the definition of monads and graded monads, and discuss how they relate to each other. To fix the notation, we recall some basic definitions:

Definition 2.1. A *functor* $T : \mathbf{Set} \to \mathbf{Set}$ consists of a set TX for each set X and a function $Tf : TX \to TY$ for each $f : X \to Y$ such that $T\mathrm{id}_X = \mathrm{id}_{TX}$ and $T(f' \circ f) = Tf' \circ Tf$. We write $T' \cdot T$ for composition of functors, and Id for the identity functor.

A *natural transformation* $\alpha : T \Rightarrow T'$ between two functors is a set-indexed family of functions $\alpha_X : TX \to T'X$ such that $\alpha_Y \circ Tf = T'f \circ \alpha_X$ for each $f : X \to Y$.

Instead of List f, we will often write map f, like a Haskell programmer would write. We will also often skip the index of a natural transformation if it is known from the context or is not important.

Definition 2.2. A *monad* (T, η, μ) consists of a functor $T : \mathbf{Set} \to \mathbf{Set}$ and two natural transformations, the *unit* $\eta : \mathrm{Id} \Rightarrow T$ and the *multiplication* $\mu : T \cdot T \Rightarrow T$, such that the monad laws hold:

$$\mu_X \circ \eta_{TX} = \mathrm{id}_{TX} \qquad \text{(left unit)}$$
$$\mu_X \circ T\eta_X = \mathrm{id}_{TX} \qquad \text{(right unit)}$$
$$\mu_X \circ T\mu_X = \mu_X \circ \mu_{TX} \qquad \text{(associativity)}$$

As was observed by Manes [14], an equivalent definition of monad can be obtained by replacing the multiplication μ with a *Kleisli extension* operation $(-)^*$. If (T, η, μ) is a monad, then for $f : X \to TY$ we define $f^* : TX \to TY$ to be $\mu_Y \circ Tf$. (In Haskell, $f^* \, t$ is written $t \gg\!= f$.)

Example 2.3. The usual list monad is (List, [−], concat), where List X is the set of all (finite, possibly-empty) lists over X, the unit [−] sends $\mathrm{x} \in X$ to the singleton list [x], and the multiplication concat sends a list of lists $\mathrm{xss} \in \mathrm{List}(\mathrm{List}\, X)$ to its concatenation $\mathrm{concat\ xss} \in \mathrm{List}\, X$. Restricting to non-empty lists gives us the usual non-empty list monad $(\mathrm{List}_{+}, [-], \mathrm{concat})$.

Graded monads are similar to monads. The primary difference is that the functor T is replaced with a *graded functor*.

Definition 2.4. If $(\mathcal{G}, \leq)$ is a preordered set, then a *$\mathcal{G}$-graded functor* T consists of a functor $T_g : \mathbf{Set} \to \mathbf{Set}$ for each $g \in \mathcal{G}$ and a natural transformation $T_{g \leq g'} : T_g \Rightarrow T_{g'}$ for each $g, g' \in \mathcal{G}$ satisfying $g \leq g'$ such that $T_{g \leq g} = \mathrm{id}_{T_g}$ and $T_{g \leq g''} = T_{g' \leq g''} \circ T_{g \leq g'}$ whenever $g \leq g' \leq g''$.

In addition to the preorder $\leq$, we need the set $\mathcal{G}$ of grades to come with a unit and a multiplication, corresponding to η and μ.

Definition 2.5. A *preordered monoid* $(\mathcal{G}, \leq, 1, \cdot)$ consists of a monoid $(\mathcal{G}, 1, \cdot)$ and a preorder $\leq$ on $\mathcal{G}$, such that $\cdot$ is monotone (if $g_1 \leq g_1'$ and $g_2 \leq g_2'$ then $g_1 \cdot g_2 \leq g_1' \cdot g_2'$).

The definition of graded monad is as follows. We restrict ourselves to considering a few concrete examples of graded monads, so for our purposes the data of the graded monad (the first half of the definition) is the most important part.

Definition 2.6. If $(\mathcal{G}, \leq, 1, \cdot)$ is a preordered monoid, then a *$\mathcal{G}$-graded monad* (T, η, μ) consists of a $\mathcal{G}$-graded functor T and a natural transformation $\eta : \mathrm{Id} \Rightarrow T_1$, and a natural transformation $\mu_{g,g'} : T_g \cdot T_{g'} \Rightarrow T_{g \cdot g'}$ for each $g, g' \in \mathcal{G}$ such that

- μ is natural in g, g':

$$T_{g_1 \cdot g' \leq g_2 \cdot g', X} \circ \mu_{g_1, g', X} = \mu_{g_2, g', X} \circ T_{g_1 \leq g_2, T_{g'} X}$$
$$T_{g \cdot g_1' \leq g \cdot g_2', X} \circ \mu_{g, g_1', X} = \mu_{g, g_2', X} \circ T_g T_{g_1' \leq g_2', X}$$

- The monad laws hold:

$$\mu_{1,g,X} \circ \eta_{T_g X} = \mathrm{id}_{T_g X} \quad \text{(left unit)}$$
$$\mu_{g,1,X} \circ T_g \eta_X = \mathrm{id}_{T_g X} \quad \text{(right unit)}$$
$$\mu_{g,g'\cdot g'',X} \circ T_g\, \mu_{g',g'',X} = \mu_{g\cdot g',g'',X} \circ \mu_{g,g',T_{g''}X} \quad \text{(associativity)}$$

Example 2.7. Let **1** be the trivial preordered monoid with one element. Functors are exactly **1**-graded functors, and monads are exactly **1**-graded monads.

Example 2.8. The set of natural numbers forms a preordered monoid $(\mathbb{N}, \leq, 1, \cdot)$ with the usual ordering and multiplication. We grade the usual (possibly-empty) list monad by this preordered monoid, with the grades $n \in \mathbb{N}$ representing upper bounds on lengths of lists. For $n \in \mathbb{N}$, define

$$\mathrm{List}_{\leq n} X = \{\mathrm{xs} \in \mathrm{List}\, X \mid |\mathrm{xs}| \leq n\}$$

where $|\mathrm{xs}|$ is the length of xs. The natural transformations $T_{m\leq n}$ are the inclusions $\mathrm{List}_{\leq m} X \subseteq \mathrm{List}_{\leq n} X$. The unit $\eta_X : X \to \mathrm{List}_{\leq 1} X$ is the singleton operation $[-]$, and the multiplication $\mu_{n,m,X} : \mathrm{List}_{\leq n}(\mathrm{List}_{\leq m} X) \to \mathrm{List}_{\leq n\cdot m} X$ is concat.

We also consider the list monad graded by the exact lengths of the lists (instead of upper bounds). In this case, the grades are elements of the preordered monoid $(\mathbb{N}, =, 1, \cdot)$ (the same as before, but with the discrete order), and the functors are now

$$\mathrm{List}_{=n} X = \{\mathrm{xs} \in \mathrm{List}\, X \mid |\mathrm{xs}| = n\}$$

Both of these examples also have non-empty variants, with natural numbers replaced by positive integers.

Now we come to the notion of a *degrading* of a graded monad T. These are essentially monads $\hat{T}$ with natural transformations $\lambda_g : T_g \Rightarrow \hat{T}$ compatible with the structure of T.

Definition 2.9. A *degrading* of a $\mathcal{G}$-graded monad (T, η, μ) consists of a monad $(\hat{T}, \hat{\eta}, \hat{\mu})$ and a natural transformation $\lambda_g : T_g \Rightarrow \hat{T}$ for each $g \in \mathcal{G}$ such that

$$\lambda_{g',X} \circ T_{g\leq g',X} = \lambda_{g,X} \qquad \hat{\eta}_X = \lambda_{1,X} \circ \eta_X$$
$$\hat{\mu}_X \circ \lambda_{g,\hat{T}X} \circ T_g \lambda_{g',X} = \lambda_{g\cdot g',X} \circ \mu_{g,g',X}$$

The definition of shallow degrading requires the pair $(\hat{T}, \lambda)$ to be the *colimit* of the underlying graded functor T.

Definition 2.10. Suppose that T is a $\mathcal{G}$-graded functor. A pair $(\hat{T}, \lambda)$ of a functor $\hat{T} : \mathbf{Set} \to \mathbf{Set}$ and a $\mathcal{G}$-indexed family of natural transformations $\lambda_g : T_g \Rightarrow \hat{T}$ such that $\lambda_{g'} \circ T_{g\leq g'} = \lambda_g$, is called the *colimit* of T if, for any other such pair (S, λ^S), there is a unique natural transformation $h : \hat{T} \Rightarrow S$ such that $\lambda^S_g = h \circ \lambda_g$.

When it exists, the colimit of T is unique (up to isomorphism). Moreover, in our setting (in **Set**), the colimit always exists: it can be constructed by defining $\hat{T}X = (\sum_{g\in\mathcal{G}} T_g X)/\equiv$, where $\equiv$ is a suitable equivalence relation. We do not give this construction in general, because our examples have simpler explicit descriptions. For both of our two graded monads of possibly-empty lists, the colimit is List together with the inclusions $\mathrm{List}_{\leq n} X \subseteq \mathrm{List} X$ or $\mathrm{List}_{=n} X \subseteq \mathrm{List} X$. For non-empty lists, the colimit is List_+ with the corresponding inclusions.

Definition 2.11. Suppose that (T, η, μ) is a $\mathcal{G}$-graded monad, and let $(\hat{T}, \lambda)$ be the colimit of T. This colimit forms a *shallow degrading* of (T, η, μ) if there are natural transformations $\hat{\eta}$ and $\hat{\mu}$ such that $(\hat{T}, \hat{\eta}, \hat{\mu})$ is a monad and the equations in the definition of degrading are satisfied.

We are particularly interested in the case when, for a given graded monad, such a shallow degrading is *unique*, but note that in general the colimit might form a degrading in more than one way, or it might fail to form a degrading at all.

If a shallow degrading $(\hat{T}, \hat{\eta}, \hat{\mu}, \lambda)$ has the additional property that, for any other degrading $(S, \eta^S, \mu^S, \lambda^S)$, the unique natural transformation $h : \hat{T} \Rightarrow S$ from the definition of colimit preserves the monad structure in the sense that

$$\eta^S_X = h_X \circ \hat{\eta}_X \qquad \mu^S_X \circ h_{SX} \circ \hat{T} h_X = h_X \circ \hat{\mu}_X$$

(in which case it is necessarily a unique shallow degrading), then it is an *algebraic Kan extension* in the sense alluded to in the introduction. This is equivalent to having a shallow degrading that is also *initial* according to Definition 6.1 below in Section 6.

3 MONADS ON List

3.1 Monads to degrade $\mathrm{List}_{\leq}$ and $\mathrm{List}_{=}$

We first consider degrading the two graded monads $\mathrm{List}_{\leq}$ and $\mathrm{List}_{=}$ of (possibly-empty) lists. In both cases, the unit is singleton, the multiplication is concatenation, and the colimit of the underlying functor is List. We therefore ask whether the ordinary list monad (List, $[-]$, concat) gives the unique shallow degrading. For $\mathrm{List}_{\leq}$, this is indeed the case:

Proposition 3.1. *The unique shallow degrading of* $(\mathrm{List}_{\leq}, [-], \mathrm{concat})$ *is the ordinary list monad* (List, $[-]$, concat) *together with the inclusions* $\mathrm{List}_{\leq n} \subseteq \mathrm{List}$.

Proof. It is easy to see that this data form a shallow degrading. The interesting part is uniqueness of the unit and multiplication.

Suppose that $\hat{\eta}$ and $\hat{\mu}$ make the colimit into a degrading. Then $\hat{\eta}$ must be singleton because $\hat{\eta}\,\mathrm{x} = \lambda_1[\mathrm{x}] = [\mathrm{x}]$, where the first equality is from the definition of degrading. For the multiplication, consider an arbitrary list of lists

$$\mathrm{xss} = [\mathrm{xs}_1, \ldots, \mathrm{xs}_n] \in \mathrm{List}(\mathrm{List} X)$$

and define $m = \max_i |\mathrm{xs}_i|$. Then $\mathrm{xss} \in \mathrm{List}_{\leq n}(\mathrm{List}_{\leq m} X)$, and we have

$$\hat{\mu}\,\mathrm{xss} = \hat{\mu}(\lambda_n(\mathrm{List}_{\leq n}\, \lambda_m\, \mathrm{xss})) = \lambda_{n\cdot m}(\mathrm{concat}\,\mathrm{xss}) = \mathrm{concat}\,\mathrm{xss}$$

□

This proof relies on the fact that if $\mathrm{xss} \in \mathrm{List}(\mathrm{List} X)$ then $\mathrm{xss} \in \mathrm{List}_{\leq n}(\mathrm{List}_{\leq m} X)$ for some m, n, which implies that any multiplication in the degrading can be written in terms of a multiplication in the graded monad.

Consider now the graded monad $\mathrm{List}_{=}$. If a monad $(\mathrm{List}, \eta, \mu)$ forms a degrading with the inclusions $\mathrm{List}_{=n} X \subseteq \mathrm{List} X$, then the unit is uniquely determined: we must have $\eta\,\mathrm{x} = [\mathrm{x}]$. Regarding the multiplication, we can apply the same trick as above for some lists of lists $\mathrm{xss} \in \mathrm{List}(\mathrm{List} X)$, but not all. Say that xss is *balanced* if all of the inner lists have the same length, i.e., $\mathrm{xss} \in \mathrm{List}_{=n}(\mathrm{List}_{=m} X)$

for some n, m. Then the multiplication is uniquely determined on balanced lists:

$$\mu\,\mathrm{xss} = \mathrm{concat}\,\mathrm{xss} \qquad \text{if xss is balanced}$$

There is a unique shallow degrading of $\mathrm{List}_=$ if and only if the multiplication is uniquely determined for all lists of lists; this is to say if and only if the monad (List, [−], concat), obviously satisfying the equation above, is the only such monad structure List.

However we can define

$$\mathrm{concat}'\,\mathrm{xss} = \begin{cases} [] & \text{if exists null xss} \\ \mathrm{concat}\ \mathrm{xss} & \text{otherwise} \end{cases}$$

Now (List, [−], concat′) is a monad too. This is because the functor List is isomorphic to Maybe · List_+ and concat′ is the multiplication of the monad structure on Maybe · List_+ from the distributive law of the ordinary non-empty list monad over the maybe monad. As concat′ xss = concat xss on balanced lists, the monad (List, [−], concat′) is a degrading of $\mathrm{List}_=$. Hence we can conclude:

Proposition 3.2. *The graded monad* ($\mathrm{List}_=$, [−], concat) *does not have a unique shallow degrading.*

We have seen two multiplications on List that agree with that of $\mathrm{List}_=$. But perhaps there are more? To answer this, we take a look at other possible monad structures on List with [−] as the unit but out of curiosity also possibly with a different unit.

3.2 Monads with a nullary-binary presentation

Since List is a finitary functor, all monads with List as the underlying functor are finitary. Finitary monads can be described with (algebraic) theories, see, e.g., [14, 20].

A *presentation* is a (not necessarily finite) collection Σ of operation symbols o with finite arities $n \in \mathbb{N}$ and a (not necessarily finite) collection E of equations on open terms made of these operation symbols. An *algebra* of a presentation is a set with operations interpreting the operation symbols so that the equations hold. An *(algebraic) theory* is an equivalence class of presentations: two presentations are identified if they have "the same" algebras, i.e., there is an isomorphism between the respective categories of algebras that preserves the underlying sets. Alternatively, two presentations are identified if they have isomorphic clones, where a *clone* of a presentation has as operation symbols all open terms of the presentation and as equations all derivable equations between them. Yet alternatively, two presentations are identified if their *free algebras* define isomorphic monads.

The *free algebra* of a presentation (Σ, E) on a set of X is the set $T_{\Sigma,E}X = (T_\Sigma X)/E$ of all terms made of operation symbols from Σ and variables drawn from X, quotiented modulo the least congruence given by the equations of E. A n-ary operation symbol o is interpreted as the function sending any given terms $(t_1, \ldots, t_n)$ to the term $o(t_1, \ldots, t_n)$. The corresponding monad (T, η, μ) has $T_{\Sigma,E}$ as the underlying functor T, the inclusion $X \subseteq TX$ of variables among terms as η_X and flattening of terms over terms over X into terms over X as the multiplication μ_X.

For example, the presentation (Σ, E) given by one nullary operation ε, one binary operation $\cdot$ and three equations

$$\varepsilon \cdot x = x \qquad x \cdot \varepsilon = x \qquad x \cdot (y \cdot z) = (x \cdot y) \cdot z$$

is a presentation of the theory of monoids. There are other presentations, e.g., we could add another operation $\cdot'$ and an equation $x \cdot' y = y \cdot x$, but that buys us nothing. The corresponding monad has as TX the carrier of the free monoid on X, i.e., the set of terms made of ε, $\cdot$ and variables drawn from X, quotiented by the above equations. This set is isomorphic to $\mathrm{List}\,X$, with [] interpreting ε and ++ interpreting $\cdot$. A variable x is interpreted as $\eta\,\mathrm{x} = [\mathrm{x}]$.

Given a monad (T, η, μ), the clone of the corresponding theory can be worked out as follows. Its operations of arity n are given by all natural transformations $o_X : (TX)^n \to TX$ such that

$$\mu_X\,(o_{TX}\,(t_1, \ldots t_n)) = o_X\,(\mu_X\,t_1, \ldots, \mu_X\,t_n)$$

The equations are given by all valid equalities between these functions. A finite presentation can only afford to use a finite subset of these functions as basic operations and the rest must be definable from those as terms. A useful fact is that natural transformations $o_X : (TX)^n \to TX$ agreeing with μ in the above way are in a bijection with natural transformations $f_X : X^n \to TX$ (with no conditions on them) via

$$o_X\,(t_1, \ldots, t_n) = \mu_X\,(f_{TX}\,(t_1, \ldots, t_n)$$

and

$$f_X\,(\mathrm{x}_1, \ldots, \mathrm{x}_n) = o_X\,(\eta_X\,\mathrm{x}_1, \ldots, \eta_X\,\mathrm{x}_n)$$

This last fact helps us in the context of list monads. The only natural transformations $X^n \to \mathrm{List}\,X$ are functions f of the form $f(\mathrm{x}_1, \ldots, \mathrm{x}_n) = [\mathrm{y}_1, \ldots, \mathrm{y}_m]$ where $\{\mathrm{y}_1, \ldots, \mathrm{y}_m\} \subseteq \{\mathrm{x}_1, \ldots, \mathrm{x}_n\}$. But they are all obtained from functions

$$\mathrm{list}_n(\mathrm{x}_1, \ldots, \mathrm{x}_n) = [\mathrm{x}_1, \ldots, \mathrm{x}_n]$$

using projections and substitution, so if we are looking for a presentation for a given list monad with a particular multiplication μ, we only need to consider the functions $\mathrm{list}^\sharp_n : (\mathrm{List}\,X)^n \to \mathrm{List}\,X$,

$$\mathrm{list}^\sharp_n(\mathrm{xs}_1, \ldots, \mathrm{xs}_n) = \mu\,(\mathrm{list}_n\,(\mathrm{xs}_1, \ldots, \mathrm{xs}_n)) = \mu\,[\mathrm{xs}_1, \ldots, \mathrm{xs}_n]$$

as candidate operations. In fact, in the monoid theory example, ε and $\cdot$ arise in this way for $n = 0$ and $n = 2$: we have

$$\varepsilon = \mathrm{list}^\sharp_0 = \mathrm{concat}\,[] = []$$

$$\mathrm{xs} \cdot \mathrm{ys} = \mathrm{list}^\sharp_2(\mathrm{xs}, \mathrm{ys}) = \mathrm{concat}\,[\mathrm{xs}, \mathrm{ys}] = \mathrm{xs} +\!\!+ \mathrm{ys}$$

Terms $t \in T_\Sigma X$ identified up to the equivalence given by E are particularly nice to work with if, in every equivalence class, there is a unique representative in a syntactically recognizable format, a term in *normal form*.

For the theory of monoids, such a normal form format exists. Terms in normal form (or, in short, normal forms) are defined as terms of one of the two forms ε or $x_1 \cdot (\ldots (x_{n-1} \cdot x_n) \ldots)$ for $n \geq 1$. Every term is equivalent to precisely one term in this form. Such terms denote lists $\mathrm{list}_0 = []$ and

$$\begin{aligned} &\mathrm{list}_2\,(\eta\,\mathrm{x}_1, \ldots (\mathrm{list}_2\,(\eta\,\mathrm{x}_{n-1}, \eta\,\mathrm{x}_n) \ldots) \\ &= [\mathrm{x}_1] +\!\!+ (\ldots ([\mathrm{x}_{n-1}] +\!\!+ [\mathrm{x}_n]) \ldots) \\ &= [\mathrm{x}_1, \ldots, \mathrm{x}_{n-1}, \mathrm{x}_n] \end{aligned}$$

which precisely enumerate all list forms. Note that this correspondence depends on η and μ (the latter defining $\varepsilon = \mathrm{list}^\sharp_0$ and $\cdot = \mathrm{list}^\sharp_2$ via list_0 and list_2) and works thanks to the specific $\eta = [-]$ and

Equations	**Multiplication** (μ xss = ...)
$\varepsilon \cdot x = x$ $x \cdot \varepsilon = x$ $(x \cdot y) \cdot z = x \cdot (y \cdot z)$	concat xss
$\varepsilon \cdot x = \varepsilon$ $x \cdot \varepsilon = \varepsilon$ $(x \cdot y) \cdot z = x \cdot (y \cdot z)$	concat′ xss = {[] if exists null xss; concat xss otherwise}
$\varepsilon \cdot x = \varepsilon$ $x \cdot \varepsilon = \varepsilon$ $(x \cdot y) \cdot z = x \cdot (y \cdot (x \cdot z))$	[] if null xss or exists null xss concat (map palindromise (init xss)) ++ last xss otherwise
$\varepsilon \cdot x = \varepsilon$ $x \cdot \varepsilon = \varepsilon$ $(x \cdot y) \cdot z = \varepsilon$	[] [] if null xss or exists null xss else if exists (not ∘ sglt) (init xss) map head (init xss) ++ last xss otherwise
$\varepsilon \cdot x = x$ $x \cdot \varepsilon = \varepsilon$ $(x \cdot y) \cdot z = y \cdot z$	[] [] if null xss else if null (last xss) concat (map safeLast (init xss)) ++ last xss otherwise
$\varepsilon \cdot x = \varepsilon$ $x \cdot \varepsilon = x^{n+2}$ $(x \cdot y) \cdot z = x \cdot y$	[] replicateLast (n + 1) (map head (takeWhile sglt (init xss))) map head (takeWhile sglt (init xss)) ++ head (dropWhile sglt (init xss) ++ [last xss]) if null xss else if null (head (dropWhile sglt (init xss) ++[last xss])) otherwise
$\varepsilon \cdot x = \varepsilon$ $x \cdot \varepsilon = x^{n+2}$ $(x \cdot y) \cdot z = x^{m+2}$	[] replicateLast (n + 1) (map head (takeWhile sglt (init xss))) map head (takeWhile sglt (init xss)) ++ replicate (m + 2) (head (head (dropWhile sglt (init xss)))) map head (init xss) ++ last xss if null xss else if null (head (dropWhile sglt (init xss) ++[last xss])) else if exists (not ∘ sglt) (init xss) or null (last xss) otherwise
$\varepsilon \cdot x = \varepsilon$ $x \cdot \varepsilon = x^{n+2}$ $(x \cdot y) \cdot z = \varepsilon$	[] replicateLast (n + 1) (map head (takeWhile sglt (init xss))) map head (init xss) ++ last xss if null xss else if exists (not ∘ sglt) (init xss) or null (last xss) otherwise

sglt xs = {False if null xs; null (tail xs) otherwise} safeLast xs = {[] if null xs; [last xs] otherwise} replicateLast n xs = {[] if null xs; xs ++ replicate n (last xs) otherwise}

palindromise xs = xs ++ reverse (init xs)

Figure 1: Examples of monads on List **with unit** $[-]$ **from theories presentable with** ε **and** $\cdot$

μ = concat, which are the unit and multiplication of the free monoid monad.

Now, it turns out that not only the theory for the monad with multiplication concat can be presented with two operations $\varepsilon = \text{list}_0^\sharp$ and $\varepsilon = \text{list}_2^\sharp$ with lists as normal forms, but the theory for the monad with multiplication concat′ can also be presented similarly. However, the equations governing ε and $\cdot$ differ. In the case of concat, they were the equations of a monoid. For concat′, they are

$$\varepsilon \cdot x = \varepsilon \qquad x \cdot \varepsilon = \varepsilon \qquad (x \cdot y) \cdot z = x \cdot (y \cdot z)$$

which are the equations of a semigroup-with-zero. The normal forms are lists, so the set List X is the carrier of not only the free monoid on X but also of the free semigroup-with-zero on X.

Can there be other monad structures on List whose theory is presentable with ε and $\cdot$ and has these normal forms?

To answer this question, we notice that a term in this signature is not in normal form if and only if it contains a subterm of one of the shapes $\varepsilon \cdot t$, $t \cdot \varepsilon$ and $(t \cdot t') \cdot t''$. So in our presentation we need three equations of the form

$$\varepsilon \cdot x = rhsL \qquad x \cdot \varepsilon = rhsR \qquad (x \cdot y) \cdot z = rhsA$$

A sufficient condition for any term in a theory presentation like this to have a unique normal form is that the equation system, read as a term rewrite system, is weakly normalizing and confluent. If, instead of just weak normalization, we can show strong normalization, then confluence can be concluded from local confluence,

which is established by checking all possible overlaps of redexes (critical pairs). Establishing strong normalization is harder and generally needs to be done on an ad hoc basis by showing that some kind of rank on terms decreases at any rewrite step. Some simple examples of such ranks are the size of the term or the number of redexes in the term, but often one needs far more sophisticated ranks.

Trying different combinations of right-hand sides (w.l.o.g. only right-hand sides that are in normal form) for the above three equations for strong normalization and local confluence, one can thus try to produce more theories with presentations with our normal form format and the unique normal forms property. Each of them yields a monad structure on List.

We performed this exercise on a number of choices, namely we checked all combinations of right-hand sides from the options

$$\begin{aligned} &\mathit{rhsL} \in \{\varepsilon, x\} \\ &\mathit{rhsR} \in \{\varepsilon, x, x^{n+2}\} \\ &\mathit{rhsA} \in \{\varepsilon, x, y, x^{m+2}, y \cdot y, x \cdot y, y \cdot x, x \cdot z, y \cdot z, x \cdot (y \cdot z), \\ &\qquad x \cdot (x \cdot z), x \cdot (y \cdot (x \cdot z)) \text{ and a few more}\} \end{aligned}$$

The result is in Figure 1. The table lists those combinations of (*rhsL*, *rhsR*, *rhsA*) that are locally confluent as term rewrite systems; those not shown have a critical pair that leads to at least two different normal forms. All combinations shown are strongly normalizing.

We found the candidate good combinations by quickchecking [5] the following parameterised candidate multiplication $\mu_{\mathit{rhsL}, \mathit{rhsR}, \mathit{rhsA}}$.

$$\begin{aligned} &\mu_{\mathit{rhsL}, \mathit{rhsR}, \mathit{rhsA}}\ \mathrm{xss} = \mu\,\mathrm{xss}\ \text{where} \\ &\quad \varepsilon = \mu\,[\,] \\ &\quad \mathrm{xs} \cdot \mathrm{ys} = \mu\,[\mathrm{xs}, \mathrm{ys}] \\ &\quad \mathrm{xs}^1 = \mathrm{xs} \\ &\quad \mathrm{xs}^{n+1} = \mathrm{xs} \cdot \mathrm{xs}^n \\ &\quad \mu\,[\,] = [\,] \\ &\quad \mu\,[\mathrm{xs}] = \mathrm{xs} \\ &\quad \mu\,([\,] :: \mathrm{xss}) = \text{let } x = \mu\,\mathrm{xss} \text{ in } \mathit{rhsL} \\ &\quad \mu\,([\mathrm{x}] :: \mathrm{yss}) = \text{case } \mu\,\mathrm{yss} \text{ of} \\ &\qquad\qquad \begin{cases} [\,] \mapsto \text{let } x = \eta\,\mathrm{x} \text{ in } \mathit{rhsR} \\ \mathrm{ys} \mapsto \mathrm{x} :: \mathrm{ys} \end{cases} \\ &\quad \mu\,((\mathrm{x} :: \mathrm{ys}) :: \mathrm{zss}) = \text{let } x = \eta\,\mathrm{x}, y = \mathrm{ys}, z = \mu\,\mathrm{zss} \text{ in } \mathit{rhsA} \end{aligned}$$

For bad values of the parameter (*rhsL*, *rhsR*, *rhsA*), the monad laws fail (or the candidate multiplication does not always terminate).

We also learned the non-recursive definitions of μ for the good cases of Figure 1 by testing $\mu_{\mathit{rhsL}, \mathit{rhsR}, \mathit{rhsA}}$ on different arguments.

We illustrate the method on the theory $(x, \varepsilon, y \cdot z)$. It is strongly terminating since every reduction step decreases the size of the term. Here are the checks of 2 critical pairs of the total 8:

$$\underline{(\varepsilon \cdot y)} \cdot \varepsilon \to y \cdot \varepsilon \to \varepsilon \qquad \underline{(\varepsilon \cdot y) \cdot \varepsilon} \to \varepsilon$$
$$\underline{((x \cdot y) \cdot z)} \cdot w \to (y \cdot z) \cdot w \to z \cdot w \qquad \underline{((x \cdot y) \cdot z) \cdot w} \to z \cdot w$$

Beyond the two good theories that we already knew, we discovered six more with this method. The theories $(\varepsilon, x^{n+2}, \mathit{rhsA})$ yield multiplications that both duplicate and delete elements. The theory $(\varepsilon, \varepsilon, x \cdot (y \cdot (x \cdot z)))$ has the multiplication both duplicating and permuting elements.

The theories with $\mathit{rhsL} = x^{n+2}$ are parameterised by a natural number n. This means that we have infinitely many monad structures on List. And not just that, infinitely many of such monads have their theory presentable with ε and $\cdot$ and have this presentation admitting a particular normal form format.

Proposition 3.3. *All theories with operations ε and $\cdot$ from Figure 1 define monads* $(\mathrm{List}, [-], \mu)$ *via the same particular normal form format. These theories are infinitely many.*

We do not know whether our table is complete. Further options for *rhsA* might give further good combinations. In particular, we do not know if there is a multiplication that permutes but does not duplicate or a multiplication that duplicates but does not delete.

In all of the above, we only considered theories presented with two operations ε, $\cdot$, with particular normal forms and with the monad unit $[-]$. Not every monad, not even with unit $[-]$ has to be presentable in this way.

3.3 A monad with no finite presentation

Here is a different monad structure on List, still with $[-]$ as the unit. As the multiplication we choose a new variation of concat:

$$\mathrm{concat}''\ \mathrm{xss} = \begin{cases} \mathrm{concat}\ \mathrm{xss} & \text{if sglt xss or all sglt xss} \\ [\,] & \text{otherwise} \end{cases}$$

We already know that, for any monad $(\mathrm{List}, \eta, \mu)$, every presentation of its theory can be reduced to one using a subset of the operations $\{\mathrm{list}^\sharp_n \mid n \in \mathbb{N}\}$ where $\mathrm{list}^\sharp_n(\mathrm{xs}_0, \ldots, \mathrm{xs}_n) = \mu\,[\mathrm{xs}_0, \ldots, \mathrm{xs}_n]$. This modified presentation will be finite if the given presentation is. Therefore, for the theory to admit a finite presentation, there has to be some $k \in \mathbb{N}$ such that the operations $\mathrm{list}^\sharp_n$ for $n < k$ suffice.

Now, for the monad $(\mathrm{List}, [-], \mathrm{concat}'')$, however we choose k, terms over the signature $\{\mathrm{list}^\sharp_n \mid n \le k\}$ can only represent lists of length at most k. This is because $\mathrm{list}^\sharp_n(\mathrm{xs}_0, \ldots, \mathrm{xs}_n)$ is non-empty only if $\mathrm{xs}_0, \ldots, \mathrm{xs}_n$ are all singletons in which case we get a list of length n. So we need all operations $\{\mathrm{list}^\sharp_n \mid n \in \mathbb{N}\}$ to present this monad.

So not every monad on List has its theory finitely presentable.

Proposition 3.4. $(\mathrm{List}, [-], \mathrm{concat}'')$ *is a monad with no finite presentation.*

3.4 Monads with a different unit

We have seen a number of choices for multiplication for List, but the unit was always $[-]$. Since $\eta_X : X \to \mathrm{List}\,X$ has to be natural, we must have $\eta\,\mathrm{x} = \mathrm{replicate}\,e\,\mathrm{x}$ for some $e \in \mathbb{N}$.

It is clear that we cannot choose $e = 0$ since then $\mu \circ \eta = \mathrm{id}$ cannot hold. But we do not know if it is possible to choose $e \ge 2$.

3.5 Monads isomorphic via reverse

The functor List is isomorphic to itself via the natural transformation reverse. This isomorphism lifts to monad structures on List.

Specifically, if $(\mathrm{List}, \eta, \mu)$ is a monad, then so is also $(\mathrm{List}, \eta, \mu^R)$ where μ^R is defined by

$$\mu^R = \mathrm{reverse} \circ \mu \circ \mathrm{List\,reverse} \circ \mathrm{reverse}$$

The proof of this is easy after noting that $\eta = \text{reverse} \circ \eta$. Furthermore, the monad with μ is isomorphic to the monad with μ^R since the natural transformation reverse is a monad morphism (it commutes with the unit and the two multiplications).

Proposition 3.5. *For any monad* (List, η, μ)*, the data* $(\text{List}, \eta, \mu^R)$ *form a monad that is isomorphic (as a monad) to the given one via the natural isomorphism* $\text{reverse}_X : \text{List}\, X \to \text{List}\, X$*, which is a monad morphism.*

All monads we have presented above are from distinct isomorphism classes. No two of them are isomorphic, neither via reverse nor via any other monad morphism.

4 MONADS ON List_+

4.1 Monads to degrade $\text{List}_{+\,\leq}$ and $\text{List}_{+=}$

We now turn to the non-empty variants of the graded monads $\text{List}_{\leq}$ and $\text{List}_{=}$.

For $\text{List}_{\leq}$, restricting to non-empty lists makes little difference: we can show that the colimit of the graded functor $\text{List}_{+\,\leq}$ is List_+ and that $(\text{List}_+, [-], \text{concat})$ is the unique shallow degrading using the same proof as in Proposition 3.1.

Proposition 4.1. *The unique shallow degrading of* $(\text{List}_{+\,\leq}, [-], \text{concat})$ *is the ordinary nonempty list monad* $(\text{List}_+, [-], \text{concat})$ *with the inclusions* $\text{List}_{+\,\leq n} \subseteq \text{List}_+$.

For $\text{List}_{=}$, however, the monad we use to show that there are multiple shallow degradings in the possibly-empty case differs from the ordinary list monad only when there is an empty list. We cannot use the same example for $\text{List}_{+=}$. In fact, it turns out that the ordinary non-empty list monad *is* the unique shallow degrading of $\text{List}_{+=}$. We prove this in Section 5 by showing that

$$\mu\, \text{xss} = \text{concat}\, \text{xss} \qquad \text{if xss is balanced}$$

uniquely determines μ.

But first we turn to a more basic question. Given any non-empty list monad $(\text{List}_+, \eta, \mu)$, what can we know about η and μ? Of course, every monad on List whose multiplication does not yield the empty list for arguments that do not contain empty lists can be restricted to a monad on List_+. If two monads on List differ in their multiplications only when the empty list is involved, the monad structures obtained this way on List_+ are identified, as in the case of monads with μ given by concat and concat′. However, there are monads on List_+ that do not seem to arise this way.

4.2 Monads presented with one binary operation

The theory of the ordinary non-empty list monad $(\text{List}_+, [-], \text{concat})$ is that of semigroups and is presented with one single binary operation $\cdot$ and the associativity equation $(x \cdot y) \cdot z = x \cdot (y \cdot z)$. This theory admits a normal form format in which the only terms in normal form are $x_1 \cdot (\ldots \cdot (x_{n-1} \cdot x_n) \ldots)$ for $n \geq 1$, corresponding to $[\text{x}_1, \ldots, \text{x}_n]$, which are all forms that a non-empty list can take. Interpreting $\cdot$ as an operation on non-empty lists (as opposed to equivalence classes of terms), we have $\text{xs} \cdot \text{ys} = \text{list}_2^\sharp\, (\text{xs}, \text{ys}) = \text{xs} +\!\!+ \text{ys}$.

We can carry out the same project we executed for list monads. The result is in Figure 2. Compared to Figure 1 for possibly-empty list monads, there is one entirely new equation that works, namely $(x \cdot y) \cdot z = x \cdot z$. The corresponding monad on List_+ was described and used by Neves [19] under the name discrete hybrid monad. Viewed as an operation on non-empty lists, $\cdot$ is in this case defined by $\text{xs} \cdot \text{ys} = \text{list}_2^\sharp\, (\text{xs}, \text{ys}) = \mu\, [\text{xs}, \text{ys}] = \text{head xs} :: \text{ys}$. Manes [12] christened algebras of the theory $(x \cdot y) \cdot z = x \cdot (y \cdot (x \cdot z))$ odd-palindrome algebras, but did not observe that free odd-palindrome algebras are non-empty lists.

Note that the last theory in Figure 2 is parameterised by a natural number m, and for each m it gives a different monad. This implies that there are infinitely many monads on List_+.

Proposition 4.2. *All theories with an operation* $\cdot$ *in Figure 2 define monads* $(\text{List}_+, [-], \mu)$ *via the same particular normal form format. These theories are infinitely many.*

4.3 Monads with a different unit

In the previous examples of monads on List_+, we saw different multiplications, but the unit was $[-]$. Since $\eta_X : X \to \text{List}_+ X$ is required to be natural, it must be that $\eta\, \text{x} = \text{replicate}\, e\, \text{x}$ for some $e \geq 1$.

Recall that, for List, we do not know of a monad that has $\eta\, \text{x} = [\text{x}, \text{x}]$. For List_+, there are such monads. Choose

$$\mu\, (\text{xs} :: \text{xss}) = \text{head xs} :: \text{concat}\, (\text{map tail xss})$$

Then $(\text{List}_+, \eta, \mu)$ is a monad. It has a finite presentation with a unary operation $\langle - \rangle$, a ternary operation $\langle -, -, - \rangle$, and the equations

$$\begin{gathered} x = \langle x, x, \langle x \rangle \rangle \\ \langle \langle x \rangle \rangle = \langle x \rangle \qquad \langle \langle x, y, z \rangle \rangle = \langle x \rangle \\ \langle x, y, \langle z \rangle \rangle = \langle x, y, \langle y \rangle \rangle \qquad \langle x, y, \langle z, v, w \rangle \rangle = \langle x, y, \langle y, v, w \rangle \rangle \\ \langle x, \langle y \rangle, \langle z \rangle \rangle = \langle x \rangle \qquad \langle x, \langle y \rangle, \langle z, v, w \rangle \rangle = \langle x, v, w \rangle \\ \langle x, \langle y, z, v \rangle, w \rangle = \langle x, z, \langle z, v, w \rangle \rangle \\ \langle \langle x \rangle, y, z \rangle = \langle x, y, z \rangle \qquad \langle \langle x, y, z \rangle, v, w \rangle = \langle x, v, w \rangle \end{gathered}$$

As operations on non-empty lists, $\langle - \rangle$ and $\langle -, -, - \rangle$ denote the functions $\langle \text{xs} \rangle = \text{list}_1^\sharp\, \text{xs} = \mu\, [\text{xs}] = [\text{head xs}]$ and $\langle \text{xs}, \text{ys}, \text{zs} \rangle = \text{list}_3^\sharp\, (\text{xs}, \text{ys}, \text{zs}) = \mu\, [\text{xs}, \text{ys}, \text{zs}] = \text{head xs} :: \text{tail ys} +\!\!+ \text{tail zs}$. This theory admits a normal form format where normal forms are terms $\langle x_1, x_2, \langle x_2, \ldots x_{n-1}, \langle x_{n-1}, x_n, \langle x_n \rangle \rangle \ldots \rangle \rangle$ for $n \geq 1$. They correspond to lists $\text{head}\, [\text{x}_1, \text{x}_1] :: \text{tail}\, [\text{x}_2, \text{x}_2] +\!\!+ \text{tail}\, (\text{head}\, [\text{x}_2, \text{x}_2] :: \ldots +\!\!+ \ldots :: \text{tail}\, [\text{x}_{n-1}, \text{x}_{n-1}] +\!\!+ \text{tail}\, (\text{head}\, [\text{x}_{n-1}, \text{x}_{n-1}] :: \text{tail}\, [\text{x}_n, \text{x}_n] +\!\!+ \text{tail}\, [\text{head}\, [\text{x}_n, \text{x}_n]]) \ldots) = [\text{x}_1, \ldots, \text{x}_n]$.

There is a straightforward way of mass-producing monads on List_+ with replicate 2 as the unit. Notice that $\text{List}_+ X \cong X \times \text{List}\, X$. Now Id is a functor with a single monad structure, and List is a functor with many monad structures as we have learned. The product $T_0 \times T_1$ of the underlying functors T_0, T_1 of two monads (T_0, η_0, μ_0) and (T_1, η_1, μ_1) always carries at least one canonical monad structure, which is the product of the two monads. The unit η of this monad is $\eta_X = \langle \eta_{0X}, \eta_{1X} \rangle$. In addition, there may be other systematic monad structures, like semidirect products, or ad hoc monad structures on $T_0 \times T_1$, not related in any systematic way to the monad structures provided on T_0 and T_1.

The first monad above is the product of the identity monad with the monad $(\text{List}, [-], \text{concat})$.

Equation	Multiplication (μ xss = ...)	
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	concat xss	
$(x \cdot y) \cdot z = x \cdot y$	map head (takeWhile sglt (init xss)) ++ head (dropWhile sglt (init xss) ++ [last xss])	
$(x \cdot y) \cdot z = x \cdot z$	map head (init xss) ++ last xss	
$(x \cdot y) \cdot z = y \cdot z$	map last (init xss) ++ last xss	
$(x \cdot y) \cdot z = x \cdot (y \cdot (x \cdot z))$	concat (map palindromise (init xss)) ++ last xss	
$(x \cdot y) \cdot z = x^{m+2}$	map head (takeWhile sglt (init xss)) ++ replicate (m + 2) (head (head (dropWhile sglt (init xss)))) map head (init xss) ++ last xss	if exists (not ∘ sglt) (init xss) otherwise

Figure 2: Examples of monads on List_+ with unit $[-]$ from theories presentable with $\cdot$

By this construction, since we have infinitely many monad structures on List with $[-]$ as the unit, we automatically have infinitely many monad structures on List_+ with replicate 2 as the unit. We also have at least one such monad structure whose theory does not admit a finite presentation.

Another simple monad on List_+ with replicate 2 as the unit has its multiplication defined by

$$\mu\,\mathrm{xss} = \begin{cases} [\mathrm{head}\ (\mathrm{head}\ \mathrm{xss})] & \text{if sglt xss or sglt (last xss)} \\ \mathrm{map}\ \mathrm{head}\ (\mathrm{init\ xss}) \\ \quad +\!\!+ \mathrm{tail}\ (\mathrm{last\ xss}) & \text{otherwise} \end{cases}$$

Its theory is presentable with three operations, for $\mathrm{list}_1^\sharp$, $\mathrm{list}_2^\sharp$ and $\mathrm{list}_3^\sharp$. This monad does not arise from the product construction. Rather, it is a modification of Neves's discrete hybrid monad on List_+.

Proposition 4.3. *Any monad structure on* List *with unit* $[-]$ *yields a monad structure on* List_+ *with unit* replicate 2 *via the product of monads construction. Not every monad structure on* List_+ *arises in this way.*

4.4 Identifying a monad by testing?

It would be nice to be able to learn a secret non-empty list monad $(\mathrm{List}_+, \eta, \mu)$, i.e., identify it as a known one by finitely testing η and μ on some arguments. For η, this takes only one test of η_X on the single element of the singleton set $X = \{0\}$. For μ, it is clear that in general one needs many tests. However, it is not clear whether a finite number of tests suffices to learn that the secret monad is some known monad on List_+, such as the ordinary non-empty list monad $(\mathrm{List}_+, [-], \mathrm{concat})$.

This turns out to be impossible already for $(\mathrm{List}_+, [-], \mathrm{concat})$. For any $p \geq 2$, define concat_p by

$$\mathrm{concat}_p\ \mathrm{xss} = \begin{cases} \mathrm{concat\ xss} & \text{if sglt xss or all sglt xss} \\ \mathrm{take}\ p\ (\mathrm{concat\ xss}) & \text{otherwise} \end{cases}$$

Proposition 4.4. $(\mathrm{List}_+, [-], \mathrm{concat}_p)$ *is a monad for each* $p \geq 2$.

The proof of this is slightly difficult, but quite long and not very interesting. We note that $p = 1$ does not give a monad.

Corollary 4.5. *For each* $p \geq 2$, *the monad* $(\mathrm{List}_+, [-], \mathrm{concat}_p)$ *satisfies* $\mathrm{concat}_p\ \mathrm{xss} = \mathrm{concat\ xss}$ *for all* X *and* $\mathrm{xss} \in \mathrm{List}_+(\mathrm{List}_+\ X)$ *such that* $|\mathrm{concat\ xss}| \leq p$ *(this holds for all* X *as soon as it holds for* $X = \{0, \ldots, p-1\}$*), but* $\mathrm{concat}_p \neq \mathrm{concat}$.

Hence we have an infinite family of monads which demonstrates that we cannot identify a secret non-empty list monad $(\mathrm{List}_+, \eta, \mu)$ as the ordinary non-empty list monad by finite testing.

4.5 Monads with no finite presentation

For List, we exhibited a monad structure with no finite presentation, namely $\eta = [-]$ and $\mu = \mathrm{concat}''$. This multiplication cannot be adapted for List_+: the function concat'' does not restrict to non-empty lists.

But similarly to the theory of the monad $(\mathrm{List}, [-], \mathrm{concat}'')$, the theory of the monad $(\mathrm{List}_+, [-], \mathrm{concat}_p)$, $p \geq 2$, cannot have a finite presentation. This is because terms made of operations $\{\mathrm{list}_n^\sharp \mid 1 \leq n \leq k\}$ can only denote non-empty lists of length at most $\max(k, p)$.

Proposition 4.6. *None of the monads* $(\mathrm{List}_+, [-], \mathrm{concat}_p)$, $p \geq 2$, *have a finite presentation.*

Thus we have in fact found infinitely many monad structures on List_+ with $[-]$ as the unit and no finite presentation of the theory.

We should notice that $\mathrm{concat}'' = \mathrm{concat}_0$. However, there are no $p \neq 0$ such that the data $(\mathrm{List}, [-], \mathrm{concat}_p)$ would give a monad. The only value of p that works for List is 0.

4.6 Some open problems

The previous and this section are far from providing a characterization of all list or non-empty list monads. We briefly mention questions we do not currently know answers for.

Multiplications that permute and duplicate. We do not have a lot of examples of multiplications that permute elements. Can elements be permuted without some elements being duplicated or deleted at the same time?

What can the unit be? The unit can only be replicate e but what can e be? Are there monads on List with $e = 2$ or monads on List_+ with $e = 3$?

More multiplications for List *from distributive laws?* We showed that, for every monad on List, one can systematically produce a monad on List_+ because $\mathrm{List}_+ = \mathrm{Id} \times \mathrm{List}$.

In the converse direction, we saw that $(\mathrm{List}, [-], \mathrm{concat}')$ arises thanks to $\mathrm{List} = \mathrm{Maybe} \cdot \mathrm{List}_+$ from a specific distributive law of $(\mathrm{List}_+, [-], \mathrm{concat})$ over the maybe monad. Now the maybe monad distributes in a unique way over any other monad. But distributive laws of other monads (T, η, μ) over the maybe monad do not always exist, and when they do, they are constructed in ad hoc ways and exploit the particularity of μ. Which other non-empty list monads distribute over the maybe monad? Perhaps they all do?

Can the multiplication be identified with testing? We proved above that checking that $\mu\,\mathrm{xss} = \mathrm{concat}\,\mathrm{xss}$ for finitely many $\mathrm{xss} \in \mathrm{List}_+(\mathrm{List}_+ X)$ for some finite set X for a secret monad $(\mathrm{List}_+, [-], \mu)$ cannot suffice to conclude that $\mu = \mathrm{concat}$.

Can we have a similar result for some other non-empty list monads? Specifically, are there monads $(\mathrm{List}_+, [-], \mu)$ for μ other than concat with the property that the data $(\mathrm{List}_+, [-], \mu_p)$ form a monad for all $p \geq 2$ where μ_p is defined by

$$\mu_p\,\mathrm{xss} = \begin{cases} \mathrm{concat}\,\mathrm{xss} & \text{if sglt xss or all sglt xss} \\ \mathrm{take}\ p\ (\mu\,\mathrm{xss}) & \text{otherwise} \end{cases}$$

This property would imply that Corollary 4.5 holds with concat replaced by μ. Therefore μ would not be identifiable by finite testing.

It is plausible (based on quickchecking) that all monads but one in Figure 2 have this property, so they are unlikely to be identifiable by finite testing.

For μ for the theory with $(x \cdot y) \cdot z = y \cdot z$, the property fails, there is a counterexample already for $p = 2$. But it seems to hold when reformulated for takeEnd p.

A further question is: do the data $(\mathrm{List}, [-], \mu_p)$ form a monad for some $p \geq 1$ for some of the monads $(\mathrm{List}, [-], \mu)$ that we know? This fails for all theories in Figure 1, except for the theory $(\varepsilon, x^{n+2}, x \cdot y)$. We have proved that this theory gives a monad for $n = 0$ and $p = 2$. It seems (based on quickchecking) that it gives a monad for all $n \in \mathbb{N}$ and $p \geq 2$.

5 SHALLOW DEGRADING OF List_+

We now prove the theorem that we announced in the previous section:

Theorem 5.1. *Let* $(\mathrm{List}_+, [-], \mu)$ *be a monad such that* $\mu\,\mathrm{xss} = \mathrm{concat}\,\mathrm{xss}$ *if* xss *is balanced. Then,* $\mu = \mathrm{concat}$.

We first introduce some terminology. For a set X, we write $\mathrm{List}_+^2 X$ for $\mathrm{List}_+(\mathrm{List}_+ X)$, and similarly for $\mathrm{List}_+^3 X$. Given $\mathrm{xss} \in \mathrm{List}_+^2 X$, we call the value $|\mathrm{concat}\,\mathrm{xss}|$ the *total length* of xss, denoted $||\mathrm{xss}||$. We call elements of the inner lists of xss *atoms*.

Lemma 5.2. *The multiplication* μ *cannot invent elements, that is, elements of* $\mu\,\mathrm{xss}$ *are atoms of* xss.

Proof. Assume $\mathrm{y} \notin X$, and $\mathrm{xss} \in \mathrm{List}_+^2(X \cup \{\mathrm{y}\})$ is such that $\mathrm{y} \in \mu\,\mathrm{xss}$, but y is not an atom of xss. Consider the naturality diagram for the inclusion $\subseteq\colon X \to X \cup \{\mathrm{y}\}$:

$$\begin{array}{ccc} \mathrm{List}_+^2 X & \xrightarrow{\mathrm{List}_+^2 \subseteq} & \mathrm{List}_+^2(X \cup \{\mathrm{y}\}) \\ \downarrow \mu & & \downarrow \mu \\ \mathrm{List}_+ X & \xrightarrow{\mathrm{List}_+ \subseteq} & \mathrm{List}_+(X \cup \{\mathrm{y}\}) \end{array}$$

Since it is also the case that $\mathrm{xss} \in \mathrm{List}_+^2 X$, we can apply the diagram above to xss. The result of the 'left–bottom' path cannot contain y, so neither can the result of the 'top–right' path, which contradicts the assumption that $\mathrm{y} \in \mu\,\mathrm{xss}$. □

Given $\mathrm{xss} = [[\mathrm{x}_1, \ldots, \mathrm{x}_{n_1}], \ldots, [\mathrm{z}_1, \ldots, \mathrm{z}_{n_k}]] \in \mathrm{List}_+^2 X$, we define its *completion*, denoted $\overline{\mathrm{xss}} \in \mathrm{List}_+^3 X$ as

$$[[\underbrace{[\mathrm{x}_1, \ldots, \mathrm{x}_1]}_{d_1}, \ldots, \underbrace{[\mathrm{x}_{n_1}, \ldots, \mathrm{x}_{n_1}]}_{d_1}], \ldots, [[\underbrace{[\mathrm{z}_1, \ldots, \mathrm{z}_1]}_{d_k}, \ldots, \underbrace{[\mathrm{z}_{n_k}, \ldots, \mathrm{z}_{n_k}]}_{d_k}]]$$

where $d_i = (\prod_{j=1}^{k} n_j)/n_i$. For example:

$$\overline{[[1,2],[3,4,5]]} = [[[1,1,1],[2,2,2]],[[3,3],[4,4],[5,5]]]$$

Completion of xss is defined specifically so that multiplying the outer two layers behaves as $\mu\,\mathrm{xss}$, while the multiplications in $\mu(\mathrm{List}_+ \mu\,\overline{\mathrm{xss}})$ work on balanced lists:

Lemma 5.3. *Given* $\mathrm{xss} \in \mathrm{List}_+^2 X$, *it is the case that*

$$\mu\,(\mathrm{List}_+ \mu\,\overline{\mathrm{xss}}) = \mathrm{concat}\,(\mathrm{List}_+\,\mathrm{concat}\,\overline{\mathrm{xss}})$$

This gives us the following two lemmata:

Lemma 5.4. *The multiplication* μ *does not delete elements, that is, the set of elements of* $\mu\,\mathrm{xss}$ *is the same as the set of atoms of* xss, *for* $\mathrm{xss} \in \mathrm{List}_+^2 X$.

Proof. Let $\mathrm{xss} \in \mathrm{List}_+^2 X$ be such that an atom x occurs in xss, but it is not an element of $\mu\,\mathrm{xss}$. Since List_+ is finitary, we can assume that all atoms of xss are distinct. Now, consider $\overline{\mathrm{xss}}$. From Lemma 5.3 we know that x occurs in $\mu\,(\mathrm{List}_+ \mu\,\overline{\mathrm{xss}})$, so (from associativity) it occurs in $\mu\,(\mu\,\overline{\mathrm{xss}})$. But, since $\mathrm{x} \notin \mu\,\mathrm{xss}$, it is the case that $[\mathrm{x}, \ldots, \mathrm{x}] \notin \mu\,\overline{\mathrm{xss}}$, and since x does not occur anywhere else in xss, it does not occur in $\mu\,(\mu\,\overline{\mathrm{xss}})$. Contradiction! □

Lemma 5.5. *The multiplication* μ *does not duplicate elements, that is, if an atom* x *occurs once in an* $\mathrm{xss} \in \mathrm{List}_+^2 X$, *it occurs at most once as an element of* $\mu\,\mathrm{xss}$.

Proof. Assume that $\mu\,\mathrm{xss}$ duplicates an atom x. Consider $\overline{\mathrm{xss}}$. From Lemma 5.3 we know that $\mu\,(\mathrm{List}_+ \mu\,\overline{\mathrm{xss}})$ contains the same number of occurrences of x as $\overline{\mathrm{xss}}$. But since $\mu\,\mathrm{xss}$ duplicates x, it duplicates the sub-sublist $[\mathrm{x}, \ldots, \mathrm{x}]$ in $\mu\,\overline{\mathrm{xss}}$, hence $\mu\,\overline{\mathrm{xss}}$ contains more occurrences of x than $\mu\,(\mathrm{List}_+ \mu\,\overline{\mathrm{xss}})$ does, which means that the outermost μ in $\mu\,(\mu\,\overline{\mathrm{xss}})$ deletes some occurrences of x. By naturality, μ therefore deletes elements on every list of lists of the same shape as $\mu(\overline{\mathrm{xss}})$, contradicting Lemma 5.4. □

Corollary 5.6. *For all* $\mathrm{xss} \in \mathrm{List}_+^2 X$, *the list* $\mu\,\mathrm{xss}$ *is a permutation of* concat xss.

It is left to prove that it is always an identity permutation. We proceed by induction on the total length of the list, ℓ. The cases when $\ell = 1$ and $\ell = 2$ are trivial, and the actual basis for the induction is $\ell = 3$:

Lemma 5.7. *For* $x, y, z \in X$,

$$\mu\,[[x, y], [z]] = [x, y, z] = \mu\,[[x], [y, z]]$$

We do not have a more insightful argument to support this lemma other than an exhaustive search via a computer program (see https://bitbucket.org/maciejpirog/degrading). Our implementation generates valid prefixes of multiplications, that is, partial functions μ defined only on lists of lists of total length n with $\mu\,xss = \mathrm{concat}\,xss$ for balanced lists xss, such that for all lists of lists of lists, μ (if defined on a particular such list) is associative. The desired result is obtained for $n = 6$, in which case there are exactly 120 valid prefixes, all satisfying Lemma 5.7.

In the following, we show that Theorem 5.1 holds also for $\ell \geq 4$, inductively assuming it holds for all lists with total length less than ℓ. This induction is not structural, and we make a heavy use of naturality. For example, if $\ell = 6$, we obtain the following equality:

$$\mu\,[[x, y], [z], [t, u], [v]] = \mu\,(\mu\,[[[x, y], [z]], [[t, u], [v]]])$$

The inner μ on the right-hand side works on a list with 4 atoms, so the equality indeed follows from the inductive hypothesis.

Lemma 5.8. *Let* $xss, tss \in \mathrm{List}(\mathrm{List}_+ X)$ *and* $ys, zs \in \mathrm{List}_+ X$ *be such that:*

1. $||xss|| + |ys| + |zs| + ||tss|| = \ell$,
2. *at least one of* xss *and* tss *is not empty,*
3. *at least one of* xss *and* tss *are not all-singletons, or* ys *and* zs *are not both singletons.*

Then, $\mu\,(xss +\!\!+ [ys +\!\!+ zs] +\!\!+ tss) = \mu\,(xss +\!\!+ [ys] +\!\!+ [zs] +\!\!+ tss)$.

Proof. We give the proof for xss and tss both non-empty. The proof is similar when either is empty.

$$\begin{aligned}
&\mu\,(xss +\!\!+ [ys +\!\!+ zs] +\!\!+ tss) \\
&= \mu\,(\mu\,[xss, [ys +\!\!+ zs], tss]) && \text{(induction)} \\
&= \mu\,[\mu\,xss, \mu\,[ys +\!\!+ zs], \mu\,tss] && \text{(associativity)} \\
&= \mu\,[\mu\,xss, ys +\!\!+ zs, \mu\,tss] && \text{(unit)} \\
&= \mu\,[\mu\,xss, \mu\,[ys, zs], \mu\,tss] && \text{(induction)} \\
&= \mu\,(\mu\,[xss, [ys, zs], tss]) && \text{(associativity)} \\
&= \mu\,(xss +\!\!+ [ys, zs] +\!\!+ tss) && \text{(induction)} \\
&= \mu\,(xss +\!\!+ [ys] +\!\!+ [zs] +\!\!+ tss) && \text{(properties of } +\!\!+\text{)}
\end{aligned}$$

Note that the proof above does not work when the assumption (2) is not true, because the middle inductive step would not have fewer atoms. It does not work when (3) is not true because the two outer inductive steps would not be on lists with fewer atoms. □

Lemma 5.9. *For* $xs, ys \in \mathrm{List}_+ X$ *such that* $|xs| + |ys| + 1 = \ell$*, it is the case that:*

$$\mu\,[xs +\!\!+ [z], ys] = \mu\,[xs, [z], ys] = \mu\,[xs, [z] +\!\!+ ys]$$

Proof. Since $\ell \geq 3$, the lists xs and ys cannot both be singletons. Hence the result follows from applying Lemma 5.8 twice. □

The two lemmata above give us that one can 'cut' an inner list into a number of inner lists, and 'glue' inner lists together to form a longer inner list – as long as the list of lists is *non-trivial*, that is, not a singleton and not all-singletons. From this, we obtain the following corollary:

Lemma 5.10. *Let* $xss, yss \in \mathrm{List}_+^2 X$ *be non-trivial lists such that* $\mathrm{concat}\,xss = \mathrm{concat}\,yss$ *and* $||xss|| = ||yss|| = \ell$*. Then,* $\mu\,xss = \mu\,yss$.

Proof. We define a relation $\sim$ on non-trivial lists of total length ℓ as the equivalence closure of the relation

$$[\ldots, xs +\!\!+ ys, \ldots] \sim [\ldots, xs, ys, \ldots]$$

From Lemmata 5.8 and 5.9 we know that if $xss \sim yss$, then $\mu\,xss = \mu\,yss$. It is left to show that $xss \sim yss$ for all non-trivial lists such that $\mathrm{concat}\,xss = \mathrm{concat}\,yss$.

We show for all non-trivial xss that if $\mathrm{concat}\,xss = xs +\!\!+ [x]$, then $xss \sim [xs, [x]]$, by induction on $|xss| > 1$:

- If $|xss| = 2$, then $xss = [ys, zs +\!\!+ [z]]$ for some ys, zs. Either zs is empty and the result is trivial, or zs is non-empty and:

  $$\begin{aligned}
  xss &= [ys, zs +\!\!+ [z]] \\
  &\sim [ys, zs, [z]] && \text{(definition of } \sim, *\text{)} \\
  &\sim [ys +\!\!+ zs, [z]] = [xs, [z]] && \text{(definition of } \sim\text{)}
  \end{aligned}$$

  For the above to be well-defined, we need to make sure that the list in $(*)$ is non-trivial, which follows from the assumption that $\ell > 3$ (so either ys or zs is not a singleton).
- Otherwise, $xss = ys :: zs :: xss'$ for non-empty ys, zs, and:

  $$\begin{aligned}
  xss &\sim (ys +\!\!+ zs) :: xss' && \text{(definition of } \sim\text{)} \\
  &\sim [xs, [x]] && \text{(induction)}
  \end{aligned}$$

Then, $\mathrm{concat}\,xss = \mathrm{concat}\,yss$ implies $\mathrm{concat}\,xss = xs +\!\!+ [x] = \mathrm{concat}\,yss$ for some xs, x, hence $xss \sim [xs, [x]] \sim yss$. □

The lemma above means that there exists a permutation π such that for all non-trivial $xss \in \mathrm{List}_+^2 X$ of total length ℓ:

$$\mu\,xss = \pi\,(\mathrm{concat}\,xss)$$

Lemma 5.11. *Assume* ℓ *is odd. Let* $xs \in \mathrm{List}_+ X$ *have* ℓ *elements. Then* $\mu\,[xs, [z]] = \pi^{-1}\,xs +\!\!+ [z]$.

Proof. Let $\ell = 2n + 1$ for a natural n. For any ys with $n + 1$ elements and zs with n elements, we have the following:

$$\begin{aligned}
&\mu\,[\pi\,(ys +\!\!+ zs), [z]] \\
&= \mu\,[\mu\,[ys, zs], [z]] && (|ys +\!\!+ zs| = \ell) \\
&= \mu\,[\mu\,[ys, zs], \mu\,[[z]]] && \text{(unit)} \\
&= \mu\,(\mu\,[[ys, zs], [[z]]]) && \text{(associativity)} \\
&= \mu\,(\mu\,[[ys], [zs, [z]]]) && \text{(Lemma 5.7)} \\
&= \mu\,[\mu\,[ys], \mu\,[zs, [z]]] && \text{(associativity)} \\
&= \mu\,[ys, \mu\,[zs, [z]]] && \text{(unit)} \\
&= \mu\,[ys, zs +\!\!+ [z]] && \text{(induction)} \\
&= ys +\!\!+ zs +\!\!+ [z] && (\mu \text{ for balanced lists})
\end{aligned}$$

In particular, we use $\mathrm{ys} +\!\!+ \mathrm{zs} = \pi^{-1}\,\mathrm{xs}$:

$$\begin{aligned}
&\mu\,[\mathrm{xs}, [\mathrm{z}]] \\
&= \mu\,[\pi\,(\pi^{-1}\,\mathrm{xs}), [\mathrm{z}]] && \text{(permutation is a bijection)} \\
&= \pi^{-1}\,\mathrm{xs} +\!\!+ [\mathrm{z}] && \text{(the above)}
\end{aligned}$$

□

Lemma 5.12. *Assume ℓ is odd. Let* $\mathrm{xs}, \mathrm{ys} \in \mathrm{List}_+X$ *be such that* $|\mathrm{xs}| = \ell$ *and* $|\mathrm{ys}| = \ell - 1$. *Then* $\mu\,[\mathrm{xs} +\!\!+ [\mathrm{z}], \mathrm{ys}] = \pi\,\mathrm{xs} +\!\!+ \pi\,([\mathrm{z}] +\!\!+ \mathrm{ys})$.

Proof.

$$\begin{aligned}
\mu\,[\mathrm{xs} +\!\!+ [\mathrm{z}], \mathrm{ys}] &= \mu\,[\pi^{-1}(\pi\,\mathrm{xs}) +\!\!+ [\mathrm{z}], \mathrm{ys}] && \text{(bijection)} \\
&= \mu\,[\mu\,[\pi\,\mathrm{xs}, [\mathrm{z}]], \mathrm{ys}] && \text{(Lemma 5.11)} \\
&= \mu\,[\mu\,[\pi\,\mathrm{xs}, [\mathrm{z}]], \mu\,[\mathrm{ys}]] && \text{(unit)} \\
&= \mu\,(\mu\,[[\pi\,\mathrm{xs}, [\mathrm{z}]], [\mathrm{ys}]]) && \text{(associativity)} \\
&= \mu\,(\mu\,[[\pi\,\mathrm{xs}], [[\mathrm{z}], \mathrm{ys}]]) && \text{(Lemma 5.7)} \\
&= \mu\,[\mu\,[\pi\,\mathrm{xs}], \mu\,[[\mathrm{z}], \mathrm{ys}]] && \text{(associativity)} \\
&= \mu\,[\mu\,[\pi\,\mathrm{xs}], \pi\,([\mathrm{z}] +\!\!+ \mathrm{ys})] && \text{(Lemma 5.10)} \\
&= \mu\,[\pi\,\mathrm{xs}, \pi\,([\mathrm{z}] +\!\!+ \mathrm{ys})] && \text{(unit)} \\
&= \pi\,\mathrm{xs} +\!\!+ \pi\,([\mathrm{z}] +\!\!+ \mathrm{ys}) && \text{(μ for balanced lists)}
\end{aligned}$$

□

Lemma 5.13. *If ℓ is odd, then π is identity.*

Proof. Let $\ell = 2n + 1$ and:

- $\mathrm{xss} = [[\mathrm{x}, \mathrm{x}'], [\mathrm{y}, \mathrm{y}'], \ldots, [\mathrm{z}, \mathrm{z}']]$ be a list with $n + 1$ elements (i.e., a list of lists with $2n + 2 = \ell + 1$ atoms),
- $\mathrm{tss} = [[\mathrm{t}, \mathrm{t}'], \ldots, [\mathrm{u}, \mathrm{u}']]$ be a list with n elements (i.e., a list of lists with $2n = \ell - 1$ atoms).

Then:

$$\begin{aligned}
&\mu\,[\mu\,\mathrm{xss}, \mu\,\mathrm{tss}] \\
&= \mu\,[[\mathrm{x}, \mathrm{x}', \mathrm{y}, \mathrm{y}' \ldots, \mathrm{z}, \mathrm{z}'], [\mathrm{t}, \mathrm{t}', \ldots, \mathrm{u}, \mathrm{u}']] && \text{(balanced lists)} \\
&= \pi\,[\mathrm{x}, \mathrm{x}', \mathrm{y}, \mathrm{y}', \ldots, \mathrm{z}] +\!\!+ \pi\,[\mathrm{z}', \mathrm{t}, \mathrm{t}', \ldots, \mathrm{u}, \mathrm{u}'] && \text{(Lemma 5.12, $*$)}
\end{aligned}$$

But:

$$\mu\,(\mu\,[\mathrm{xss}, \mathrm{tss}]) = \mu\,(\pi\,(\mathrm{xss} +\!\!+ \mathrm{tss})) \qquad \text{(Lemma 5.10, $**$)}$$

By associativity, $(*)$ and $(**)$ are equal. Since μ in $(**)$ works on a balanced list (each inner list is of length two, of the form $[\mathrm{x}, \mathrm{x}']$), $(**)$ is equal to $[\ldots, \mathrm{z}, \mathrm{z}', \ldots]$ (that is, z and z' are next to each other). But in $(*)$, z is somewhere in the first half of the list, while z' is somewhere in the second half of the list. This means that π in $(*)$ puts z as the last element of the left-hand side list, and z' as the first element of the right-hand side list, that is, preserves their original positions.

Next, note that in $(**)$, x' is right after x, which means that the left-hand side π in $(*)$ puts x' at the second position (since x is preserved as the first element). This means that the second position is also preserved. Also, in $(**)$, t' is right after t. In the right-hand side π in $(*)$, the position of t is preserved (because it is the second element on the list), so the right-hand side π in $(*)$ puts t' as the third element. This means that the third element is also preserved. The same reasoning now applies to y, y', and the rest of the list. □

Lemma 5.14. *Let* $\mathrm{xs}, \mathrm{ys} \in \mathrm{List}_+X$ *be such that* $|\mathrm{xs}| + |\mathrm{ys}| = \ell$. *Then,* $\mu\,[\mathrm{xs}, \mathrm{ys}] = \mathrm{xs} +\!\!+ \mathrm{ys}$.

Proof. Case 1: If ℓ is even, one can find two lists, zs and ts such that $|\mathrm{zs}| = |\mathrm{ts}|$ and $\mathrm{zs} +\!\!+ \mathrm{ts} = \mathrm{xs} +\!\!+ \mathrm{ys}$. Then:

$$\begin{aligned}
\mu\,[\mathrm{xs}, \mathrm{ys}] &= \mu\,[\mathrm{zs}, \mathrm{ts}] && \text{(Lemma 5.10)} \\
&= \mathrm{zs} +\!\!+ \mathrm{ts} && \text{(μ for balanced lists)} \\
&= \mathrm{xs} +\!\!+ \mathrm{ys} && \text{(assumption)}
\end{aligned}$$

Case 2: ℓ is odd. Use Lemma 5.13. □

We can finally prove the main theorem:

Proof of Theorem 5.1. To show that $\mu\,\mathrm{xss} = \mathrm{concat}\,\mathrm{xss}$, we proceed by induction on $|\mathrm{xss}|$, that is, the number of inner lists of xss. If $|\mathrm{xss}| = 1$, the theorem holds trivially from the unit law. If $|\mathrm{xss}| = 2$, the theorem follows from Lemma 5.14. If $|\mathrm{xss}| > 2$ and all inner lists are singletons, the theorem follows from the other unit law. Otherwise, $\mathrm{xss} = [\mathrm{xs}, \mathrm{ys}] +\!\!+ \mathrm{zss}$ for some xs, ys, and zss. Then:

$$\begin{aligned}
&\mu\,([\mathrm{xs}, \mathrm{ys}] +\!\!+ \mathrm{zss}) \\
&= \mu\,(\mu\,([[\mathrm{xs}, \mathrm{ys}]] +\!\!+ \mathrm{map}\,[-]\,\mathrm{zss})) && \text{(induction on ℓ)} \\
&= \mu\,([\mu\,[\mathrm{xs}, \mathrm{ys}]] +\!\!+ \mathrm{map}\,(\mu \circ [-])\,\mathrm{zss}) && \text{(associativity)} \\
&= \mu\,([\mathrm{xs} +\!\!+ \mathrm{ys}] +\!\!+ \mathrm{map}\,(\mu \circ [-])\,\mathrm{zss}) && \text{(Lemma 5.14)} \\
&= \mu\,([\mathrm{xs} +\!\!+ \mathrm{ys}] +\!\!+ \mathrm{zss}) && \text{(unit)} \\
&= \mathrm{concat}\,([\mathrm{xs} +\!\!+ \mathrm{ys}] +\!\!+ \mathrm{zss}) && \text{(induction on $|\mathrm{xss}|$)} \\
&= \mathrm{concat}\,([\mathrm{xs}, \mathrm{ys}] +\!\!+ \mathrm{zss}) && \text{(properties of concat)}
\end{aligned}$$

□

6 INITIAL DEGRADINGS

So far, we have considered a two-step degrading: first, we construct a functor via a colimit, and then we check if there is a unique extension of the graded-monad structure to a non-graded structure. While this construction has its advantages (namely, the underlying functor agrees with the intuition about existential quantification of grades), and it gave us an opportunity to study the space of monad structures on lists, we should also put it in the categorical context, to wit: whether it is characterised by some sort of a universal property. The one we inspect is the notion that deserves to be called initial degrading:

Definition 6.1. Let $(\hat{T}, \hat{\eta}, \hat{\mu}, \lambda)$ be a degrading from Definition 2.9. We say that it is *initial* if, for any other degrading $(S, \eta^S, \mu^S, \lambda^S)$, there is a unique natural transformation $h : \hat{T} \Rightarrow S$ such that the following equations hold:

$$\lambda^S_{g,X} = h_X \circ \lambda_{g,X} \qquad \eta^S_X = h_X \circ \hat{\eta}_X \qquad \mu^S_X \circ h_{SX} \circ \hat{T} h_X = h_X \circ \hat{\mu}_X$$

Unlike for a colimit (Definition 2.10), the natural transformation h is only required to exist when S forms a degrading, and is only required to be unique amongst those that preserve the monad structure. However, h *is* required to preserve the monad structure. Initial degradings are unique up to structure-preserving isomorphism.[1]

[1]For the categorically minded reader: A graded monad on **Set** is a lax monoidal functor from $\mathcal{G}$ to [**Set**, **Set**]. An initial degrading is a colimit of (T, η, μ) taken in the 2-category **MonCat** of monoidal categories, lax monoidal functors and monoidal transformations, instead of the 2-category **Cat** of categories, functors and natural transformations. An algebraic Kan extension (an algebraic colimit in our situation) is a colimit in **MonCat** that is also a colimit in **Cat**.

Definition 6.1 is an unpacked definition of a (non-graded-)monad reflection of a graded monad. In particular, one may define the category **GMnd** whose objects are pairs $(\mathcal{G}, T)$ of a preordered monoid and a $\mathcal{G}$-graded monad, and morphisms $(\mathcal{G}, T) \to (\mathcal{G}', T')$ are pairs consisting of a preordered monoid morphism $F : \mathcal{G} \to \mathcal{G}'$ and a monoidal transformation $T \Rightarrow T' \cdot F$. A non-graded monad S can be seen as an object $(\mathbf{1}, S)$, where $\mathbf{1}$ is the trivial preordered monoid with one element. Thus, we may see the category **Mnd** of monads as a full subcategory of the category **GMnd**. The initial degrading of a graded monad T is then the free object in **Mnd** with respect to the subcategory inclusion functor generated by T.

Both $\mathrm{List}_{=}$ and $\mathrm{List}_{+=}$ have initial degradings and and we construct them below. Note that, if the initial degrading is given by the colimit, i.e., is a shallow degrading, then it is the unique shallow degrading. For $\mathrm{List}_{=}$, we have shown that there are multiple shallow degradings. Thus, if an initial degrading exists, then it cannot be given by the colimit. Surprisingly, even though $\mathrm{List}_{+=}$ does have a unique shallow degrading, it does not match the initial one.

We construct the initial degrading of the graded monad $\mathrm{List}_{+=}$ of non-empty lists. The initial degrading in the possibly-empty case has an almost identical construction: just replace List_{+} with List below.

For each set X, define $\hat{T}_0 X$ inductively with two constructors, lf ("leaf") and nd ("node"), so that $\mathrm{lf}\ \mathrm{x} \in \hat{T}_0 X$ for each $\mathrm{x} \in X$ and $\mathrm{nd}\ \mathrm{ts} \in \hat{T}_0 X$ for each $\mathrm{ts} \in \mathrm{List}_{+}(\hat{T}_0 X)$. We let $\approx$ be the smallest equivalence relation on $T_0 X$ closed under the following three rules. The first is a congruence rule, the second and third are there to satisfy the definition of degrading.

$$\begin{aligned} \mathrm{nd}\,[\mathrm{t}_1, \ldots, \mathrm{t}_n] &\approx \mathrm{nd}\,[\mathrm{t}'_1, \ldots, \mathrm{t}'_n] && \text{if } \mathrm{t}_i \approx \mathrm{t}'_i \text{ for each } i \\ \mathrm{nd}\,[\mathrm{lf}\ \mathrm{x}] &\approx \mathrm{lf}\ \mathrm{x} && \text{if } \mathrm{x} \in X \\ \mathrm{nd}\,(\mathrm{List}_{+}\ \mathrm{nd}\ \mathrm{tss}) &\approx \mathrm{nd}\,(\mathrm{concat}\ \mathrm{tss}) && \text{if } \mathrm{tss} \in \mathrm{List}_{+}(\mathrm{List}_{+}(\hat{T}_0 X)) \text{ is balanced} \end{aligned}$$

We make the quotient $\hat{T}X = (\hat{T}_0 X)/{\approx}$ into a monad $(\hat{T}, \hat{\eta}, \hat{\mu})$. The unit is the singleton operation $\hat{\eta}\,\mathrm{x} = \mathrm{lf}\ \mathrm{x}$ (we implicitly take equivalence classes). The action of the functor $\hat{T}$ on functions f, and the multiplication $\hat{\mu}$, are defined inductively:

$$\begin{aligned} \hat{T} f\ (\mathrm{lf}\ \mathrm{x}) &= \mathrm{lf}\ (f\ \mathrm{x}) & \hat{T} f\ (\mathrm{nd}\ \mathrm{ts}) &= \mathrm{nd}\ (\mathrm{List}_{+}\ (\hat{T} f)\ \mathrm{ts}) \\ \hat{\mu}\ (\mathrm{lf}\ \mathrm{t}) &= \mathrm{t} & \hat{\mu}\ (\mathrm{nd}\ \mathrm{tts}) &= \mathrm{nd}\ (\mathrm{List}_{+}\ \hat{\mu}\ \mathrm{tts}) \end{aligned}$$

Checking that this is a monad is routine. To get a degrading of $\mathrm{List}_{+=}$, we also need natural transformations $\lambda_n : \mathrm{List}_{+=n} \Rightarrow \hat{T}$. These are given by $\lambda_n\,[\mathrm{x}_1, \ldots, \mathrm{x}_n] = \mathrm{nd}\,[\mathrm{lf}\ \mathrm{x}_1, \ldots, \mathrm{lf}\ \mathrm{x}_n]$. The requirements in the definition of degrading follow from the definition of $\approx$.

Theorem 6.2. *The monad $(\hat{T}, \hat{\eta}, \hat{\mu})$ together with the natural transformations λ_n defined above form the initial degrading of the graded monad $\mathrm{List}_{+=}$ of non-empty lists.*

Proof. We give the proof of initiality. Suppose that (S, η^S, μ^S) forms another degrading with $\lambda^S_n : \mathrm{List}_{+=n} \Rightarrow S$. We need to show that there is a unique natural transformation $h : \hat{T} \Rightarrow S$ that commutes with the structure of the degradings. For uniqueness of h on nd we have

$$\begin{aligned} h\ (\mathrm{nd}\ \mathrm{ts}) &= h\ (\hat{\mu}(\lambda_{|\mathrm{ts}|}\mathrm{ts})) && \text{(definitions)} \\ &= \mu^S(h(\hat{T}h(\lambda_{|\mathrm{ts}|}\mathrm{ts}))) && (h \text{ commutes with } \mu) \\ &= \mu^S(h(\lambda_{|\mathrm{ts}|}(\mathrm{List}_{+}\ h\ \mathrm{ts}))) && (\lambda \text{ natural}) \\ &= \mu^S(\lambda^S_{|\mathrm{ts}|}(\mathrm{List}_{+}\ h\ \mathrm{ts})) && (h \text{ commutes with } \lambda) \end{aligned}$$

Similar reasoning can be applied to lf, showing that h must be given inductively by

$$h\ (\mathrm{lf}\ \mathrm{x}) = \eta^S \mathrm{x} \qquad h\ (\mathrm{nd}\ \mathrm{ts}) = \mu^S(\lambda^S_{|\mathrm{ts}|}(\mathrm{List}_{+}\ h\ \mathrm{ts}))$$

This is well-defined (with respect to $\approx$) because the monad (S, η^S, μ^S) is required to agree with the graded monad on balanced lists. In particular, for $\mathrm{tss} \in \mathrm{List}_{+=n}(\mathrm{List}_{+=m}(\hat{T}X))$ balanced, we have

$$\begin{aligned} & h\ (\mathrm{nd}\ (\mathrm{List}_{+}\ \mathrm{nd}\ \mathrm{tss})) \\ &= \mu^S(\lambda^S_n(\mathrm{List}_{+=n}(\mu^S \circ \lambda^S_m \circ \mathrm{List}_{+=m} h)\mathrm{tss})) && \text{(definitions)} \\ &= \mu^S(S\mu^S(\lambda^S_n(\mathrm{List}_{+=n}(\lambda^S_m \circ \mathrm{List}_{+=m} h)\ \mathrm{tss})))) && (\lambda^S \text{ natural}) \\ &= \mu^S(\mu^S(\lambda^S_n(\mathrm{List}_{+=n}(\lambda^S_m \circ \mathrm{List}_{+=m} h)\ \mathrm{tss})))) && (\mu^S \text{ associative}) \\ &= \mu^S(\lambda^S_{n\cdot m}(\mathrm{concat}(\mathrm{List}_{+=n}(\mathrm{List}_{+=m} h)\ \mathrm{tss})) && \text{(balanced)} \\ &= h\ (\mathrm{nd}\ (\mathrm{concat}\ \mathrm{tss})) && \text{(definitions)} \end{aligned}$$

It is easy to check the other two rules of $\approx$, and that h commutes with the structure of the degradings. □

For non-empty lists, we have proved that there is only one monad structure on List_{+} that agrees with the graded monad. In the remainder of this section we show that, even though the monad structure is unique, the colimit of $\mathrm{List}_{+=}$ still does not form the initial degrading.

By uniqueness, if List_{+} does form a degrading, then the unit is singleton and the multiplication is concatenation. The natural transformation $h : \hat{T} \Rightarrow \mathrm{List}_{+}$ in this case is given by

$$h\ (\mathrm{lf}\ \mathrm{x}) = [\mathrm{x}] \qquad h\ (\mathrm{nd}\ \mathrm{ts}) = \mathrm{concat}\ (\mathrm{List}_{+}\ h\ \mathrm{ts})$$

We use a notion of *balancedness* for elements of $\hat{T}_0 X$, defined inductively by the following rules:

- $\mathrm{lf}\ \mathrm{x}$ is balanced for all $\mathrm{x} \in X$.
- $\mathrm{nd}\ \mathrm{ts}$ is balanced if every element of $\mathrm{ts} \in \mathrm{List}_{+}(\hat{T}_0 X)$ is balanced and $\mathrm{List}_{+}\ h\,\mathrm{ts} \in \mathrm{List}_{+}(\mathrm{List}_{+} X)$ is a balanced list of lists.

This notion of balancedness is crucially preserved by $\approx$. The proof relies on the following lemma about balancedness for lists of lists.

Lemma 6.3. *Suppose that $\mathrm{tss} \in \mathrm{List}_{+=n}(\mathrm{List}_{+=m}(\hat{T}X))$. The following are equivalent:*

1. *$\mathrm{List}_{+=n\cdot m}\ h\ (\mathrm{concat}\ \mathrm{tss}) \in \mathrm{List}_{+=n\cdot m}(\mathrm{List}_{+} X)$ is balanced.*
2. *There exists $k > 0$ such that*

$$\mathrm{List}_{+=n}\ (\mathrm{List}_{+=m} h)\ \mathrm{tss} \in \mathrm{List}_{+=n}(\mathrm{List}_{+=m}(\mathrm{List}_{+=k} X))$$

3. *$\mathrm{List}_{+=n}\ (\mathrm{concat} \circ \mathrm{List}_{+=m} h)\ \mathrm{tss} \in \mathrm{List}_{+=n}(\mathrm{List}_{+} X)$ is balanced, and $\mathrm{List}_{+=m}\ h\,\mathrm{ts} \in \mathrm{List}_{+=m}(\mathrm{List}_{+} X)$ is balanced for every $\mathrm{ts} \in \mathrm{tss}$.*

Proof. Showing that (1) and (2) are equivalent is simple. To show (2) implies (3), note that $\mathrm{List}_{+=m}\, h\, \mathrm{ts} \in \mathrm{List}_{+=m}(\mathrm{List}_{+=k}\, X)$ is balanced for each $\mathrm{ts} \in \mathrm{tss}$, hence so is

$$\mathrm{List}_{+=n}(\mathrm{concat} \circ \mathrm{List}_{+=m} h)\ \mathrm{tss} \in \mathrm{List}_{+=n}(\mathrm{List}_{+=m\cdot k}\, X)$$

To show that (3) implies (2), let $\mathrm{tss} = [\mathrm{ts}_1, \ldots, \mathrm{ts}_n]$. Then for each i we have $\mathrm{concat}\,(\mathrm{List}_{+=n}\, h\, \mathrm{ts}_i) \in \mathrm{List}_{+=m\cdot|\mathrm{ts}_i|} X$, and balancedness of $\mathrm{List}_{+=n}(\mathrm{concat} \circ \mathrm{List}_{+=m} h)\ \mathrm{tss}$ implies $m \cdot |\mathrm{ts}_1| = \cdots = m \cdot |\mathrm{ts}_n|$, so $|\mathrm{ts}_1| = \cdots = |\mathrm{ts}_n|$ is a suitable k. □

Lemma 6.4. *Suppose that* $\mathrm{t} \approx \mathrm{t}'$ *for* $\mathrm{t}, \mathrm{t}' \in \hat{T}_0 X$. *Then* t *is balanced if and only if* t' *is balanced.*

Proof. By induction on the derivation of $\mathrm{t} \approx \mathrm{t}'$. For $\mathrm{lf\,x} \approx \mathrm{nd}\,[\mathrm{lf\,x}]$, both sides are balanced. For $\mathrm{nd}\,(\mathrm{concat\ tss}) \approx \mathrm{nd}\,(\mathrm{List}_+\ \mathrm{nd\ tss})$ where $\mathrm{tss} \in \mathrm{List}_+(\mathrm{List}_+(\hat{T}_0\, X))$ is balanced, note that by definition nd (concat tss) is balanced if and only if both every atom of tss is balanced, and $\mathrm{List}_+\ h\ (\mathrm{concat\ tss}) \in \mathrm{List}_+(\mathrm{List}_+ X)$ is balanced. The latter is equivalent to (3) in Lemma 6.3, and hence together they are equivalent to balancedness of $\mathrm{nd}\,(\mathrm{List}_+\ \mathrm{nd\ tss})$. Finally, if $\mathrm{t}_i \approx \mathrm{t}'_i$ for all i, then $\mathrm{nd}[\mathrm{t}_1, \ldots, \mathrm{t}_n]$ is balanced if and only if each t_i is balanced and the lengths of $h\,\mathrm{t}_i$ are all the same. Since $\mathrm{t}_i \approx \mathrm{t}'_i$ implies $h\,\mathrm{t}_i = h\,\mathrm{t}'_i$, this is equivalent to balancedness of $\mathrm{nd}\,[\mathrm{t}_1, \ldots, \mathrm{t}_n]$. □

However, the natural transformation h identifies balanced and unbalanced elements of TX, and this implies List_+ cannot form the initial degrading.

Theorem 6.5. *The functor* List_+*, together with the inclusions* $\lambda_n : \mathrm{List}_{+=n} \Rightarrow \mathrm{List}_+$*, does not form an initial degrading of* $\mathrm{List}_{+=}$.

Proof. Suppose that List_+ does form an initial degrading in this way. The unit must be singleton and the multiplication must be concatenation by Theorem 5.1, and hence we have a natural transformation $h : \hat{T} \Rightarrow \mathrm{List}_+$ defined as above. Since $\hat{T}$ forms a degrading, there is a natural transformation $h^{-1} : \mathrm{List}_+ \Rightarrow \hat{T}$ that commutes with the structure of the two degradings. But $\hat{T}$ is also initial by Theorem 6.2, so h^{-1} must be the inverse of h.

To determine h^{-1}, note that

$$[\mathrm{x}_1, \ldots, \mathrm{x}_n] = h\,(\mathrm{nd}\,[\mathrm{lf\,x}_1, \ldots, \mathrm{lf\,x}_n])$$

Since h^{-1} is the inverse of h, this implies

$$h^{-1}[\mathrm{x}_1, \ldots, \mathrm{x}_n] = \mathrm{nd}\,[\mathrm{lf\,x}_1, \ldots, \mathrm{lf\,x}_n]$$

Finally, we obtain a contradiction by considering an unbalanced element of $\hat{T}X$, such as $\mathrm{nd}\,[\mathrm{lf\,x}, \mathrm{nd}[\mathrm{lf\,y}, \mathrm{lf\,z}]]$. We have

$$\begin{aligned}\mathrm{nd}\,[\mathrm{lf\,x}, \mathrm{nd}\,[\mathrm{lf\,y}, \mathrm{lf\,z}]] &= h^{-1}(h\,(\mathrm{nd}\,[\mathrm{lf\,x}, \mathrm{nd}\,[\mathrm{lf\,y}, \mathrm{lf\,z}]])) \\ &= h^{-1}[\mathrm{x}, \mathrm{y}, \mathrm{z}] \\ &= \mathrm{nd}\,[\mathrm{lf\,x}, \mathrm{lf\,y}, \mathrm{lf\,z}]\end{aligned}$$

but the latter is balanced, contradicting Lemma 6.4. □

7 DISCUSSION

As our results suggest, associating monads to graded monads as envisaged by Fritz and Perrone [7] is a subtle endeavour. On the one hand, canonical degradings given by universal constructions are not necessarily the most natural or useful ones. On the other hand, it also seems that the existence and canonicity of degradings can be shown only on a case-by-case basis hinging on the very specific properties of a given graded monad, rather than generalities.

There are some well-studied problems related to our results on list monads, for example, completing an object mapping to obtain a functor, that is, inventing its action on morphisms [2–4]. However, to our knowledge there has been little work on the totalities of possible monad structures on set endofunctors, either in general or in specific cases. Some examples are the work by Manes on extending set functors to (approximations of) monads [12], results by Manes and Mulry [13], Klin and Salamanca [10], Zwart and Marsden [26] about (non-)existence of distributive laws between various monads, or Uustalu's [24] characterisation of monad structures on container functors (in the sense of Abbott *et al.* [1]). Other results of more combinatorial nature include enumerating preorders on monads by Katsumata and Sato [9, 22], or describing monads for theories with "polynomial" Cayley representations by Piróg *et al.* [21].

In this paper, we give a number of constructions of monads on List and List_+, but invent few specific properties that would apply to all monad structures on them. Hence, an interesting open question is if a complete classification of all monad structures on List or List_+ is possible. The only result that gives us some constraints on possible structures is Theorem 5.1. We managed to prove it using elementary means, but its assumption about coherence with the usual structure in the balanced case is rather strong. It seems that to obtain more universal results, one needs to apply more heavyweight combinatorial machinery.

Acknowledgments. We thank Flavien Breuvart, Jeremy Gibbons, Renato Neves, Paolo Perrone and Exequiel Rivas for discussions and our anonymous reviewers for many useful remarks. This research was supported by the Icelandic Research Fund project grant no. 196323-052. T.U. was also supported by the Estonian Ministry of Education and Research institutional research grant no. IUT33-13.

REFERENCES

[1] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. 2005. Containers: Constructing Strictly Positive Types. *Theoretical Computer Science* 342, 1 (2005), 3–27. https://doi.org/10.1016/j.tcs.2005.06.002

[2] Artur Barkhudaryan. 2003. Endofunctors of Set Determined by Their Object Map. *Applied Categorical Structures* 11, 6 (2003), 507–520. https://doi.org/10.1023/a:1026177620363

[3] Artur Barkhudaryan, Robert El Bashir, and Věra Trnková. 2003. Endofunctors of Set and Cardinalities. *Cahiers de topologie et géométrie différentielle catégoriques* 44, 3 (2003), 217–239. http://www.numdam.org/item/CTGDC_2003__44_3_217_0/

[4] Daniela Cancila, Furio Honsell, and Marina Lenisa. 2006. Functors Determined by Values on Objects. *Electronic Notes in Theoretical Computer Science* 158 (2006), 151–169. https://doi.org/10.1016/j.entcs.2006.04.009

[5] Koen Claessen and John Hughes. 2000. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming, ICFP '00, Montreal, Canada, Sept. 18–21, 2000*, Martin Odersky and Philip Wadler (Eds.). ACM, New York, 268–279. https://doi.org/10.1145/351240.351266

[6] Ulrich Dorsch, Stefan Milius, and Lutz Schröder. 2019. Graded Monads and Graded Logics for the Linear Time - Branching Time Spectrum. In *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, Wan Fokkink and Rob van Glabbeek (Eds.). Leibniz Int. Proceedings in Informatics, Vol. 140. Dagstuhl Publishing, Saarbrücken/Wadern. https://doi.org/10.4230/lipics.concur.2019.36

[7] Tobias Fritz and Paolo Perrone. 2018. A Criterion for Kan Extensions of Lax Monoidal Functors. arXiv eprint. arXiv:1809.10481 [math.CT]

[8] Shin-ya Katsumata. 2014. Parametric Effect Monads and Semantics of Effect Systems. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, New York, 633–646. https://doi.org/10.1145/2535838.2535846

[9] Shin-ya Katsumata and Tetsuya Sato. 2013. Preorders on Monads and Coalgebraic Simulations. In *Foundations of Software Science and Computation Structures - 16th International Conference, FOSSACS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, Frank Pfenning (Ed.). Lecture Notes in Computer Science, Vol. 7794. Springer, Cham, 145–160. https://doi.org/10.1007/978-3-642-37075-5_10

[10] Bartek Klin and Julian Salamanca. 2018. Iterated Covariant Powerset Is Not a Monad. *Electronic Notes in Theoretical Computer Science* 341 (2018), 261–276. https://doi.org/10.1016/j.entcs.2018.11.013

[11] Seerp Roald Koudenburg. 2015. Algebraic Kan Extensions in Double Categories. *Theory and Applications of Categories* 30, 5 (2015), 86–146. http://www.tac.mta.ca/tac/volumes/30/5/30-05abs.html

[12] Ernie Manes. 2003. Monads of Sets. In *Handbook of Algebra, Vol. 3*, Michiel Hazewinkel (Ed.). Elsevier, Amsterdam, 67–153. https://doi.org/10.1016/s1570-7954(03)80059-1

[13] Ernie Manes and Philip S. Mulry. 2008. Monad Compositions II: Kleisli Strength. *Mathematical Structures in Computer Science* 18, 3 (2008), 613–643. https://doi.org/10.1017/s0960129508006695

[14] Ernest G. Manes. 1976. *Algebraic Theories*. Graduate Texts in Mathematics, Vol. 26. Springer, New York. https://doi.org/10.1007/978-1-4612-9860-1

[15] Paul-André Melliès. 2017. The Parametric Continuation Monad. *Mathematical Structures in Computer Science* 27, 5 (2017), 651–680. https://doi.org/10.1017/S0960129515000328

[16] Paul-André Melliès and Nicolas Tabareau. 2008. Free Models of T-Algebraic Theories Computed as Kan Extensions. HAL eprint. https://hal.archives-ouvertes.fr/hal-00339331/

[17] Stefan Milius, Dirk Pattinson, and Lutz Schröder. 2015. Generic Trace Semantics and Graded Monads. In *6th Conference on Algebra and Coalgebra in Computer Science, CALCO 2015, June 24–26, 2015, Nijmegen, The Netherlands*, Lawrence S. Moss and Paweł Sobociński (Eds.). Leibniz Int. Proceedings in Informatics, Vol. 35. Dagstuhl Publishing, Saarbrücken/Wadern, 253–269. https://doi.org/10.4230/lipics.calco.2015.253

[18] Alan Mycroft, Dominic Orchard, and Tomas Petricek. 2016. Effect Systems Revisited—Control-Flow Algebra and Semantics. In *Semantics, Logics, and Calculi: Essays Dedicated to Hanne Riis Nielson and Flemming Nielson on the Occasion of Their 60th Birthdays*, Christian W. Probst, Chris Hankin, and René Rydhof Hansen (Eds.). Lecture Notes in Computer Science, Vol. 9560. Springer, Cham, 1–32. https://doi.org/10.1007/978-3-319-27810-0_1

[19] Renato Neves. 2018. *Hybrid Programs.* Ph.D. Dissertation. University of Minho.

[20] Maria Cristina Pedicchio and Fabrizio Rovatti. 2003. Algebraic Categories. In *Categorical Foundations: Special Topics in Topology, Algebra, and Sheaf Theory*, Maria Cristina Pedicchio and Walter Tholen (Eds.). Encyclopedia of Mathematics and Its Applications, Vol. 97. Cambridge University Press, Cambridge, 269–310. https://doi.org/10.1017/cbo9781107340985.009

[21] Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. 2019. Equational Theories and Monads from Polynomial Cayley Representations. In *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, Mikołaj Bojańczyk and Alex Simpson (Eds.). Lecture Notes in Computer Science, Vol. 11425. Springer, Cham, 453–469. https://doi.org/10.1007/978-3-030-17127-8_26

[22] Tetsuya Sato. 2014. Identifying All Preorders on the Subdistribution Monad. *Electronic Notes in Theoretical Computer Science* 308 (2014), 309–327. https://doi.org/10.1016/j.entcs.2014.10.017

[23] Alexander Smirnov. 2008. Graded Monads and Rings of Polynomials. *Journal of Mathematical Sciences* 151 (2008), 3032–3051. https://doi.org/10.1007/s10958-008-9013-7

[24] Tarmo Uustalu. 2017. Container Combinatorics: Monads and Lax Monoidal Functors. In *Topics in Theoretical Computer Science - Second IFIP WG 1.8 International Conference, TTCS 2017, Tehran, Iran, September 12-14, 2017, Proceedings*, Mohammad Reza Mousavi and Jiří Sgall (Eds.). Lecture Notes in Computer Science, Vol. 10608. Springer, Cham, 91–105. https://doi.org/10.1007/978-3-319-68953-1_8

[25] Mark Weber. 2016. Algebraic Kan Extensions along Morphisms of Internal Algebra Classifiers. *Tbilisi Mathematical Journal* 9, 1 (2016), 65–142. https://doi.org/10.1515/tmj-2016-0006

[26] Maaike Zwart and Dan Marsden. 2019. No-Go Theorems for Distributive Laws. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*. IEEE, Los Alamitos, CA, Article 8785707, 13 pages. https://doi.org/10.1109/lics.2019.8785707