Counting Monads on Lists

Dylan McDermott, Reykjavík University
Maciej Piróg, Standard Chartered
Tarmo Uustalu, Reykjavík University and Tallinn University of Technology

1 Introduction

Monads are one of the fundamental concepts in category theory. They play an important role in semantics of programming languages, process algebra, and the practice of programming. A monad can be concisely defined as an endofunctor equipped with a structure of a monoid in the category of endofunctors and natural transformations. (We unpack these notions into our concrete setting of lists on the category **Set** of sets and functions in Section 2.) One rarely studied aspect of monads are combinatorial questions, such as: given an endofunctor, how many such monoid structures are there and what do they look like?

While we are not aware of many general combinatorial approaches to category theory (in **Set** or generally), some results on possible monad structures on given endofunctors have already seen applications, mostly in the study of composition of monads [1, 4], which is important in understanding how computational effects compose in programming languages. Other examples include work on the relationship of monads and graded monads [2], and explorations of equational presentation of monads [3].

In this note, we focus on the endofunctor of lists (that is, finite sequences) on the category **Set**. The most standard example of a monad on lists is formed by singleton as the unit and concatenation as the multiplication. It is used in programming and semantics to model nondeterministic computations – and is usually referred to as *the* list monad. However, as we show in a previous work [2], it is only one of infinitely many ways to turn the list endofunctor into a monad. Moreover, the picture is quite diverse: there are infinitely many list monads that are generated by a finite equational theory on a binary an nullary operation, as well as list monads that are not generated by any finite equational theory; there are monads that discard, duplicate, and shuffle the elements. Here, we settle the question of the size of the set of list monads: we prove that the set of list monads on **Set** has the cardinality of the continuum.

2 List Monads on Set

Given a set A, we define LA to be the set of all finite, possibly empty sequences (lists) of elements coming from A. We write $[x_1, \ldots, x_n]$ to enumerate the elements of a list. As a convention, for a fixed set A, we use xs, ys, etc. as variables standing for lists of type LA, and xss for $L(LA)$. We use $+\!\!+$ as concatenation of elements, that is $[x_1, \ldots, x_n] +\!\!+ [y_1, \ldots, y_k] = [x_1, \ldots, x_n, y_1, \ldots, y_k]$. For a function $f : A \to B$, we define $Lf : LA \to LB$ to be $Lf([a_1, \ldots, a_k]) = [f(a_1), \ldots, f(a_k)]$.

The direct instantiation of the definition of a monad on lists on **Set** (the category of sets and functions) can be expressed in terms of two families of functions indexed by sets: $\eta_A : A \to LA$ (the *unit*) and $\mu_A : L(LA) \to LA$ (the *multiplication*) subject to:

- $Lf(\eta_A(a)) = \eta_B(f(a))$, for all $a \in A$ and $f : A \to B$,
- $Lf(\mu_A(xss)) = \mu_B(L(Lf)(xss))$, for all $xss \in L(LA)$ and $f : A \to B$,
- $\mu_A(L\eta_A(xs)) = xs = \mu_A(\eta_{LA}(xs))$, for all $xs \in LA$,
- $\mu_A(\mu_{LA}(xss)) = \mu_A(L\mu_A(xss))$, for all $xss \in L(LA)$.

The first two laws say that η and μ are natural transformations – their actions do not depend on the concrete values of the elements of the lists. From this, we can derive a more compact characterisation: every list monad is uniquely determined by η_1 (where 1 is a singleton set) and $\mu_{\mathbb{N}}$ with the domain restricted to lists $[xs, \ldots, zs]$ such that $xs +\!\!+ \cdots +\!\!+ zs = [0, 1, \ldots, k]$ for some k, and for all elements x of $\mu_{\mathbb{N}}([xs, \ldots, zs])$, x is an element of $xs +\!\!+ \cdots +\!\!+ zs$, as long as these two functions satisfy the last two laws.

The last two laws are the properties of a monoid in the category of endofunctors. The former says that η is a neutral element of μ on both sides, that is, both when applied to each individual element and to the entire list. The latter is the associativity: if we go from $L(L(LA))$ to LA, it doesn't matter if we first multiply the inner or outer two layers. For example, the usual list monad is defined by $\eta(a) = [a]$ and $\mu([xs, \ldots, zs]) = xs +\!\!+ \cdots +\!\!+ zs$.

The characterisation above means that the number of possible units is at most $\aleph_0$, and there are no more than $|(L\mathbb{N})^{L(L\mathbb{N})}| = 2^{\aleph_0}$ possible multiplications. In Section 4, we show a construction of a family of list monads injectively indexed by subsets of the set of odd natural numbers, and so of the cardinality $2^{\aleph_0}$.

3 Warmup: Numerical Monoids

Each list monad is an infinite object, as $\mu_{\mathbb{N}}$ needs to be defined on an infinite domain. Moreover, there exist monads with multiplications that discard, shuffle, and duplicate elements [2]. So, to curb this complexity, in this note we narrow our focus to monads for which $\eta(a) = [a]$, while $\mu([xs, \ldots, zs])$ equals either $xs +\!\!+ \cdots +\!\!+ zs$ or $[\,]$ (the empty list). Thus, the multiplication is determined by a set of finite sequences of natural numbers that represent the lengths of the elements of lists for which μ is concatenation. We say that such sequences are *accepted* by μ. It remains to establish which sets of accepted sequences induce a monad.

We start with a simple example, which leads only to a countable family of monads, but which, we believe, provides useful intuitions about our setting. This candidate can be obtained by choosing a property of non-empty lists that is preserved by concatenation if all the inner lists and the outer list all satisfy the property, and "defaulting" to the empty list the moment we encounter a list that does not satisfy the property. In our setting, the preservation by multiplication becomes a form of closure under addition:

We call a set $C \subseteq \mathbb{N}$ *good* if $0 \notin C$, $1 \in C$, and for all $k \in C$ and $n_1, \ldots, n_k \in C$ it is the case that $\sum_{i=1}^{k} n_i \in C$. Examples of good sets include the singleton set $\{1\}$, the set $\{n \mid n \text{ is odd}\}$ (since 1 is odd and the sum of odd number of odd numbers is odd) and $\{1\} \cup \{n, n+1, \ldots\}$ for any $n > 1$. Every good set C induces a monad with $\eta(a) = [a]$ and

$$\begin{aligned}
\mu([xs]) &= xs \\
\mu([[x_1], \ldots, [x_n]]) &= [x_1, \ldots, x_n] \\
\mu([xs_1, \ldots, xs_k]) &= xs_1 +\!\!+ \cdots +\!\!+ xs_k && \text{if } k \in C \text{ and } |xs_i| \in C \text{ for all } i = 1, \ldots, k \\
\mu(xss) &= [\,] && \text{otherwise}
\end{aligned}$$

Clearly, different good sets induce different monads, but how many good sets are there? By C^- we denote the set $\{n - 1 \mid n \in C\}$. It turns out that C is good if and only if C^- is a numerical monoid, that is, $0 \in C^-$ and for all $k, n \in C^-$ it is the case that $k + n \in C^-$. Moreover, it is known that every numerical monoid has a finite set of generators.[1] This means that there are exactly as many numerical monoids as finite sets of natural numbers (that is, $\aleph_0$), and so the set of monads induced by good sets as above is also of the cardinality $\aleph_0$.

4 An Uncountable Family of List Monads

The construction above gives us only countably many monads, because it is in a sense too dense: the associativity of μ forces too many inputs to be accepted. To obtain uncountably many list

[1] This fact is often expressed as an exercise in basic computational complexity: the Kleene closure of any set of words over a unary alphabet is regular.

monads, we need less interaction between inputs. We obtain this by proposing a multiplication whose results never correspond to accepted sequences, whatever their inner lists might be.

Let G be a subset of the set of odd natural numbers. We define a monad with $\eta(a) = [a]$ and

$$\mu([xs]) = xs \tag{R1}$$
$$\mu([[x_1], \ldots, [x_n]]) = [x_1, \ldots, x_n] \tag{R2}$$
$$\mu([[x_1, \ldots, x_n], [y]]) = [x_1, \ldots, x_n, y] \quad \text{if } n \in G \tag{R3}$$
$$\mu(xss) = [\,] \quad \text{otherwise} \tag{R4}$$

This means that the set of accepted inputs include the trivial singleton and all-singleton lists, together with lists of the shape $xs = [[x_1, \ldots, x_n], [y]]$ for $n \in G$. Since G contains only odd numbers, the result of $\mu([[xs_1, \ldots, xs_n], [ys]])$, that is $[xs_1, \ldots, xs_n, ys]$ is not accepted. This is of course just a rough idea, the full proof that this is indeed a monad requires some more detailed case analysis to deal with the singleton cases.

Different sets G induce different monads, and since we put no further constraints on G, there are as many such monads as subsets of the set of odd natural numbers, that is, $2^{\aleph_0}$.

5 Discussion

Knowing just the cardinality of the set of list monads is not entirely satisfactory. One future direction would be to explore the structure of this set: is being a list monad a random property, or is a form of classification theorem possible? In our concatenate-or-empty setting, one can start with any set of inputs, and form a valid monad by taking the closure of this set by adding lists that need to be accepted because of the associativity of μ (such a closure always exists, because the set of all positive natural numbers is good). Then, one measure of the randomness of the structure of the set of all list monads would be decidability, e.g., if a given input is in the smallest closure of the given finite initial set.

It is also possible to find some classes of non-examples. For instance, we hypothesise (based on extensive computer search) that there are no list monads with units other than $\eta(a) = [a]$.

References

[1] Bartek Klin and Julian Salamanca. Iterated covariant powerset is not a monad. In *MFPS*, 2018. DOI 10.1016/j.entcs.2018.11.013.

[2] Dylan McDermott, Maciej Piróg, and Tarmo Uustalu. Degrading lists. In *PPDP*, 2020. DOI 10.1145/3414080.3414084.

[3] Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Equational theories and monads from polynomial Cayley representations. In *FOSSACS*, 2019. DOI 10.1007/978-3-030-17127-8_26.

[4] Maaike Zwart and Dan Marsden. No-go theorems for distributive laws. In *LICS*, 2019. DOI 10.5555/3470152.3470189.

A Proofs

In the following, we consider the candidate for a monad defined by $\eta(a) = [a]$ and μ defined by the rules (R1)–(R4) for a set $G \subseteq \mathcal{P}\{n \mid n \text{ odd}, n \geq 3\}$. We begin with the following three trivial lemmas, for which we skip the proofs:

Lemma 1. *For xss such that $|xss| \geq 3$, if at least one of the elements of xss is not a singleton, then $\mu(xss) = [\,]$.*

Lemma 2. *For all xss, it is the case that $|\mu(xss)|$ is odd only if one of the following holds:*

1. *xss is all-singleton, i.e., $xss = [[x_1], \ldots, [x_n]]$ for an odd n,*

2. *xss is a singleton, i.e.,* $xss = [[x_1, \ldots, x_n]]$ *for an odd* n.

Lemma 3. *For all xss,* $\mu(xss) = [x]$ *if and only if* $xss = [[x]]$.

Theorem 4. *The natural transformations* η *and* μ *defined as above form a list monad.*

Proof. The unit laws hold trivially due to rules (R1) and (R2). For the associativity, we consider a list of lists of lists $xsss$. We need to show that $\mu(\mu(xsss)) = \mu(L\mu(xsss))$. We proceed by case analysis of the rule that can be used to calculate $\mu(xsss)$.

- Case (R1). In this case, $xsss = [xss]$ for some xss. We assume that $\mu(xss) = xs$ for some xs. This means that:
$$\mu(\mu([xss])) = \mu(xss) = xs \qquad \text{((R1), assumption)}$$
while:
$$\mu(L\mu([xss])) = \mu([xs]) = xs \qquad \text{(assumption, (R1))}$$
- Case (R2). In this case, $xsss = [[xs_1], \ldots, [xs_n]]$ for some $xs_1, \ldots, xs_n$. We assume that $\mu([xs_1, \ldots, xs_n]) = xs$ for some xs. This means that:
$$\mu(\mu([[xs_1], \ldots, [xs_n]])) = \mu([xs_1, \ldots, xs_n]) = xs \qquad \text{((R2), assumption)}$$
while:
$$\mu(L\mu([[xs_1], \ldots, [xs_n]]) = \mu([xs_1, \ldots, xs_n]) = xs \qquad \text{((R1), assumption)}$$
- Case (R3). In this case, $xsss = [[xs_1, \ldots, xs_n], [ys]]$ for some $n \in G$. We consider three cases, depending on whether the inner-most lists are singletons:
  - Subcase: $xs_1, \ldots, xs_n, ys$ are all singletons. This means that $xsss = [[[x_1], \ldots, [x_n]], [[y]]]$ for some $x_1, \ldots, x_n$, and y. This gives us:
  $$\mu(\mu([[[x_1], \ldots, [x_n]], [[y]]])) = \mu([[x_1], \ldots, [x_n], [y]]) = [x_1, \ldots, x_n, y] \qquad \text{((R3), (R2))}$$
  while:
  $$\mu(L\mu([[[x_1], \ldots, [x_n]], [[y]]])) = \mu([[x_1, \ldots, x_n], [y]]) = [x_1, \ldots, x_n, y] \qquad \text{((R2), (R3))}$$
  - Subcase: There is a non-singleton list among $xs_1, \ldots, xs_n$. This means that:
  $$\mu(\mu([[xs_1, \ldots, xs_n], [ys]])) = \mu([xs_1, \ldots, xs_n, ys]) = [] \qquad \text{((R3), Lemma 1)}$$
  while:
  $$\mu(L\mu([[xs_1, \ldots, xs_n], [ys]])) = \mu([[], \mu([ys])]) = [] \qquad \text{(Lemma 1, (R4))}$$
  - Subcase: ys is not a singleton. This means that:
  $$\mu(\mu([[xs_1, \ldots, xs_n], [ys]])) = \mu([xs_1, \ldots, xs_n, ys]) = [] \qquad \text{((R3), Lemma 1)}$$
  while:
  $$\mu(L\mu([[xs_1, \ldots, xs_n], [ys]])) = \mu([\mu([xs_1, \ldots, xs_n]), ys]) = [] \qquad \text{((R1), (R4))}$$
- Case (R4). In this case, $\mu(xsss) = []$. We proceed by case analysis of the rule used to calculate the outer μ in $\mu(L\mu(xsss))$:
  - Subcase (R1). In this case, $L\mu(xsss) = [xss]$ for some xss, which means that $\mu(xsss)$ is a singleton. This contradicts $\mu(xsss) = []$, therefore this subcase cannot happen.
  - Subcase (R2). In this case, $L\mu(xsss) = [[x_1], \ldots, [x_n]]$ for some n. Using Lemma 3, it follows that $xsss = [[[x_1]], \ldots, [[x_n]]]$, but $\mu([[[x_1]], \ldots, [[x_n]]]) = [[x_1], \ldots, [x_n]]$ by (R2). This contradicts $\mu(xsss) = []$, therefore this subcase cannot happen.

– Subcase (R3). In this case, $L\mu(xsss) = [[x_1, \ldots, x_n], [y]]$ for some $n \in G$, and thus $xsss = [xss, yss]$ such that $\mu(xss) = [x_1, \ldots, x_n]$ and $\mu(yss) = [y]$. By Lemma 2, $xss = [[x_1], \ldots, [x_n]]$ or $xss = [[x_1, \ldots, x_n]]$. By Lemma 3, $yss = [[y]]$. We proceed by case analysis of the two possible values of xss:

  * Subsubcase: $xss = [[x_1], \ldots, [x_n]]$. In this case, $xsss = [[[x_1], \ldots, [x_n]], [[y]]]$. But $\mu([[[x_1], \ldots, [x_n]], [[y]]]) = [[x_1], \ldots, [x_n], [y]]$ by (R3), which contradicts $\mu(xsss) = [\,]$. Therefore, this subsubcase cannot happen.
  * Subsubcase: $xss = [[x_1, \ldots, x_n]]$. In this case, $xsss = [[[x_1, \ldots, x_n]], [[y]]]$. But $\mu([[[x_1, \ldots, x_n]], [[y]]]) = [[x_1, \ldots, x_n], [y]]$ by (R2), which contradicts $\mu(xsss) = [\,]$. Therefore, this subsubcase cannot happen.

– Subcase (R4). We have:

$$\mu(\mu(xsss)) = \mu([\,]) = [\,] \qquad \text{(assumption of case, (R4))}$$

while:

$$\mu(L\mu(xsss)) = [\,] \qquad \text{(assumption of subcase)}$$

□