List monads

Dylan McDermott  Maciej Piróg  Tarmo Uustalu
Outline

How many monad structures are there on the functor $\text{List} : \text{Set} \to \text{Set}$?
Outline

How many monad structures are there on the functor $\text{List} : \text{Set} \rightarrow \text{Set}$?

1. How many powerset monads are there?
2. How many list monads are there?
3. Degradings of graded monads
A monad consists of

- an endofunctor $T$;
- a natural transformation $\eta_X : X \to TX$;
- a natural transformation $\mu_X : T(TX) \to TX$;

such that the monad laws hold:

$$TX \xrightarrow{\eta_{TX}} T(TX) \xrightarrow{T\eta_X} T(TX) \xrightarrow{T\mu_X} T(TX)$$

$$T(TX) \xrightarrow{\mu_X} TX \xrightarrow{T\mu_X} T(TX) \xrightarrow{T\mu_X} T(TX)$$

Example: the usual list monad is given by

$$T = \text{List} : \text{Set} \to \text{Set}$$

$$X \mapsto \text{set of finite possibly-empty lists over } X$$

$$f \mapsto \lambda[x_1, \ldots, x_n]. [fx_1, \ldots, fx_n]$$

$$\eta_X = \lambda x. [x] \quad \mu_X = \lambda[xs_1, \ldots, xs_n]. xs_1 ++ \cdots ++ xs_n$$
The usual list monad is given by

\[ T = \text{List} : \text{Set} \to \text{Set} \]

\[ X \mapsto \text{set of finite possibly-empty lists over } X \]

\[ f \mapsto \lambda [x_1, \ldots, x_n]. [fx_1, \ldots, fx_n] \]

\[ \eta_X = \lambda x. [x] \quad \mu_X = \lambda [xs_1, \ldots, xs_n]. xs_1 ++ \cdots ++ xs_n \]

Question: if the functor \( T : \text{Set} \to \text{Set} \) is one of

\begin{itemize}
  \item List – finite lists
  \item \( \text{List}_+ \) – nonempty finite lists
  \item \( \mathcal{P} \) – subsets
  \item \( \mathcal{P}_+ \) – nonempty subsets
  \item \ldots
\end{itemize}

are there any monad structures other than the usual one?
All of the powerset monads

The covariant powerset functor $\mathcal{P} : \text{Set} \to \text{Set}$, with

$$\mathcal{P} f = \lambda S. \{ fx | x \in S \}$$

forms a monad in exactly two ways.

The unit $\eta_X : X \to \mathcal{P}X$ is always

$$\eta_X = \lambda x. \{ x \}$$

The multiplication $\mu_X : \mathcal{P}(\mathcal{P}X) \to \mathcal{P}X$ is one of

$$\mu_X = \lambda S. \bigcup S \quad \mu_X = \lambda S. \begin{cases} \emptyset & \text{if } \emptyset \in S \\ \bigcup S & \text{otherwise} \end{cases}$$

The nonempty powerset functor $\mathcal{P}_+ : \text{Set} \to \text{Set}$ forms a monad in exactly one way.
All of the powerset monads

There are exactly two natural transformations \( \alpha_X : \mathcal{P}X \to \mathcal{P}X \).

**Proof**

1. For every \( S \in \mathcal{P}X \), \( \alpha_X S \subseteq S \), because

\[
\begin{array}{c}
\mathcal{P}S \xrightarrow{\alpha_S} \mathcal{P}S \\
\mathcal{P} \subseteq & \quad & \mathcal{P} \subseteq \\
\mathcal{P}X \xrightarrow{\alpha_X} \mathcal{P}X
\end{array}
\]

2. For every \( S \), if \( \alpha_X S \) is non-empty then \( \alpha_X S = S \).

3. For every \( S \), either \( \alpha_X S = \emptyset \) or \( \alpha_X S = S \), because

\[
\mathcal{P} \subseteq \mathcal{P} \subseteq 1 \mathcal{P} \langle \rangle \xrightarrow{\alpha} \mathcal{P} \langle \rangle \xrightarrow{1} \mathcal{P} X
\]

So \( \alpha \) is one of \( \alpha_X = \lambda S \).
All of the powerset monads

There are exactly two natural transformations \( \alpha_X : \mathcal{P}X \to \mathcal{P}X \).

Proof

1. For every \( S \in \mathcal{P}X \), \( \alpha_X S \subseteq S \).

2. For every \( S \), if \( \alpha_X S \) is non-empty then \( \alpha_X S = S \): if \( x \in \alpha_X S \subseteq S \) then for every \( y \in S \),

\[
\begin{array}{ccc}
\mathcal{P}X & \xrightarrow{\alpha_X} & \mathcal{P}X \\
\mathcal{P}\text{swap}_{x,y} \downarrow & & \downarrow \mathcal{P}\text{swap}_{x,y} \\
\mathcal{P}X & \xrightarrow{\alpha_X} & \mathcal{P}X
\end{array}
\]

so \( y \in \mathcal{P}\text{swap}_{x,y}(\alpha_X S) = \alpha_X S \).
All of the powerset monads

There are exactly two natural transformations $\alpha_X : \mathcal{P}X \to \mathcal{P}X$.

Proof

1. For every $S \in \mathcal{P}X$, $\alpha_X S \subseteq S$.
2. For every $S$, if $\alpha_X S$ is non-empty then $\alpha_X S = S$.
3. For every $S$, either $\alpha_X S = \emptyset$ or $\alpha_X S = S$. 
All of the powerset monads

There are exactly two natural transformations $\alpha_X : \mathcal{P}X \to \mathcal{P}X$.

Proof

1. For every $S \in \mathcal{P}X$, $\alpha_X S \subseteq S$.
2. For every $S$, if $\alpha_X S$ is non-empty then $\alpha_X S = S$.
3. For every $S$, either $\alpha_X S = \emptyset$ or $\alpha_X S = S$.
4. Either $\alpha_X S = \emptyset$ for every $S$, or $\alpha_X S = S$ for every $S$, because

\[
\begin{array}{ccc}
\mathcal{P}X & \xrightarrow{\alpha_X} & \mathcal{P}X \\
\mathcal{P}\langle \rangle & \downarrow & \downarrow \mathcal{P}\langle \rangle \\
\mathcal{P}1 & \xrightarrow{\alpha_1} & \mathcal{P}1 \\
\end{array}
\]

So $\alpha$ is one of

$$\alpha_X = \lambda S. \emptyset \quad \alpha_X = \lambda S. S$$
All of the powerset monads

By similar proofs, the unit and multiplication

\[ \eta_X : X \to \mathcal{P}X \quad \mu_X : \mathcal{P}(\mathcal{P}X) \to \mathcal{P}X \]

are completely determined by

\[ \eta_1 : 1 \to \mathcal{P}1 \quad \mu_1 : \mathcal{P}(\mathcal{P}1) \to \mathcal{P}1 \]

but only two pairs \((\eta_1, \mu_1)\) satisfy the monad laws.
Lists are harder

Natural transformations

\[ \alpha_X : \text{List}X \rightarrow \text{List}X \]

are not completely determined by

\[ \alpha_1 : \text{List}1 \rightarrow \text{List}1 \]

For example

\[ \text{id}_1 = \text{reverse}_1 \quad \text{but} \quad \text{id} \neq \text{reverse} \]
Lists are harder

Natural transformations

\[ \alpha_X : \text{List}X \rightarrow \text{List}X \]

are not completely determined by

\[ \alpha_1 : \text{List}1 \rightarrow \text{List}1 \]

They are completely determined by

\[ \alpha_2 : \text{List}2 \rightarrow \text{List}2 \]

but this doesn't seem to help much.
What we can say for certain

For $T = \text{List}$:

- $\eta_{X} x = [x, \ldots, x]$ for some $e > 0$ that doesn’t depend on $X, x$.
- If $x$ appears somewhere in $\mu_{X} \text{xs}$, then $x$ appears somewhere in $\text{xs}$.
- Every monad structure has a presentation with (maybe infinitely many) operators of finite arity. These will do when $e = 1$:

\[
(\lambda(xs_1, \ldots, xs_n). \mu_X [xs_1, \ldots, xs_n]) : (\text{List } X)^n \to \text{List } X
\]

Similar things hold for $T = \text{List}_{+}$. 
Some possibly-empty list monads

For $\eta_X = \lambda x. [x]$ we can define $\mu_X : \text{List} (\text{List} X) \rightarrow \text{List} X$ by:

$$\mu_X = \lambda xss. \text{concat} xss$$

$$\mu_X = \lambda xss. \begin{cases} [] & \text{if } [] \in xss \\ \text{concat} xss & \text{otherwise} \end{cases}$$

$$\varepsilon \cdot x = x = x \cdot \varepsilon$$

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

$$\varepsilon \cdot x = \varepsilon = x \cdot \varepsilon$$

$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$
Some possibly-empty list monads

The monad presented by $\varepsilon : 1$ and $(\cdot) : 2$ with equations

\[
\varepsilon \cdot x = \varepsilon = x \cdot \varepsilon
\]
\[
(x \cdot y) \cdot z = x \cdot (y \cdot (x \cdot z))
\]

has $\text{List} : \text{Set} \to \text{Set}$ as the underlying functor, and $\eta_X = \lambda x. [x]$. 
### Equations

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon \cdot x = x$</td>
<td></td>
</tr>
<tr>
<td>$x \cdot \varepsilon = x$</td>
<td>concat $x$s</td>
</tr>
<tr>
<td>$(x \cdot y) \cdot z = x \cdot (y \cdot z)$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon \cdot \varepsilon = \varepsilon$</td>
<td>concat' $x$s =</td>
</tr>
<tr>
<td>$x \cdot \varepsilon = \varepsilon$</td>
<td>concat $x$s</td>
</tr>
<tr>
<td>$(x \cdot y) \cdot z = x \cdot (y \cdot z)$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon \cdot \varepsilon = \varepsilon$</td>
<td>[]</td>
</tr>
<tr>
<td>$\varepsilon \cdot \varepsilon = \varepsilon$</td>
<td>concat (map palindrome (init $x$s)) ++ last $x$s</td>
</tr>
<tr>
<td>$(x \cdot y) \cdot z = x \cdot (y \cdot z)$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon \cdot x = \varepsilon$</td>
<td>[]</td>
</tr>
<tr>
<td>$x \cdot \varepsilon = \varepsilon$</td>
<td>[]</td>
</tr>
<tr>
<td>$(x \cdot y) \cdot z = \varepsilon$</td>
<td>concat (safeLast (init $x$s)) ++ last $x$s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon \cdot x = \varepsilon$</td>
<td>[]</td>
</tr>
<tr>
<td>$x \cdot \varepsilon = \varepsilon$</td>
<td>[]</td>
</tr>
<tr>
<td>$(x \cdot y) \cdot z = y \cdot z$</td>
<td>concat (map safeLast (init $x$s)) ++ last $x$s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon \cdot x = \varepsilon$</td>
<td>[]</td>
</tr>
<tr>
<td>$x \cdot \varepsilon = x^{n+2}$</td>
<td>replicateLast ($n + 1$)</td>
</tr>
<tr>
<td>$(x \cdot y) \cdot z = x \cdot y$</td>
<td>(map head (takeWhile sglt (init $x$s)))</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon \cdot x = \varepsilon$</td>
<td>[]</td>
</tr>
<tr>
<td>$x \cdot \varepsilon = x^{n+2}$</td>
<td>replicateLast ($n + 1$)</td>
</tr>
<tr>
<td>$(x \cdot y) \cdot z = x^{n+2}$</td>
<td>(map head (takeWhile sglt (init $x$s)))</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon \cdot x = \varepsilon$</td>
<td>[]</td>
</tr>
<tr>
<td>$x \cdot \varepsilon = x^{n+2}$</td>
<td>replicateLast ($n + 1$)</td>
</tr>
<tr>
<td>$(x \cdot y) \cdot z = \varepsilon$</td>
<td>(map head (takeWhile sglt (init $x$s)))</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon \cdot x = \varepsilon$</td>
<td>[]</td>
</tr>
<tr>
<td>$x \cdot \varepsilon = \varepsilon$</td>
<td>[]</td>
</tr>
<tr>
<td>$(x \cdot y) \cdot z = \varepsilon$</td>
<td>concat $x$s</td>
</tr>
</tbody>
</table>

### Multiplication ($\mu x s = \ldots$)

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

### Figure 1: Examples of monads on List with unit [-] from theories presentable with $\varepsilon$ and $\cdot$
Some possibly-empty list monads

For $\eta_X = \lambda x. [x]$ we can define $\mu_X : \text{List}(\text{List}X) \to \text{List}X$ by:

$$
\mu_X = \lambda xss. \begin{cases} 
[] & \text{if } xss \text{ is not a singleton} \\
\text{concat } xss & \text{and } xss \text{ contains a non-singleton} \\
\end{cases}
$$

No presentation with finitely many operators, because for fixed $p$ the algebraic operations

$$(\lambda(xs_1, \ldots, xs_n). \mu_X [xs_1, \ldots, xs_n]) : (\text{List } X)^n \to \text{List } X \quad (n \leq p)$$

generate lists of length $\leq p$. 
How many list monads are there?

Answer: infinitely many

- Can discard elements
- Can duplicate elements
- Can have no finite presentation
Some non-empty list monads

For $T = \text{List}_+$ and $\eta_X = [x]$, can define $\mu_X$ by

$$
\mu [xs_1, \ldots, xs_n] = \text{head } xs_1 :: \cdots :: \text{head } xs_{n-1} :: xs_n
$$
Some non-empty list monads

For $T = \text{List}_+$ and $\eta_X = [x]$, can define $\mu_X$ by

$$\mu_X = \begin{cases} 
\text{concat } xss & \text{if } xss \text{ is a singleton, or all-singletons} \\
\text{take 11 (concat } xss) & \text{otherwise}
\end{cases}$$

Requires infinitely many operators!
Some non-empty list monads

For $T = \text{List}_+$ and $\eta_X = [x, x]$, can define $\mu_X$ by

$$
\mu_{xss} = \text{head} (\text{head} xss) :: \text{concat} (\text{tail} (\text{List}_+ \text{ tail } xss))
$$

This arises from $\text{List}_+ \cong \text{Id} \times \text{List}$
How many non-empty list monads are there?

Answer: infinitely many

- Can discard elements
- Can duplicate elements
- Can have no finite presentation
- Can have $\eta x \neq [x]$
What is the relationship between monads and graded monads?
What is the relationship between monads and graded monads?

- Monads $T$ organize computations into sets $TX$ (e.g. $TX = \text{lists over } X$)
- Graded monads organize computations into sets $T_gX$ (e.g. $T_gX = \text{lists over } X \text{ of length } g$)
- The grades $g$ provide quantitative information (e.g. number of alternatives in a nondeterministic computation)
What is the relationship between monads and graded monads?

- Monads $T$ organize computations into sets $TX$
  (e.g. $TX = \text{lists over } X$)
- Graded monads organize computations into sets $T_gX$
  (e.g. $T_gX = \text{lists over } X \text{ of length } g$)
- The grades $g$ provide quantitative information
  (e.g. number of alternatives in a nondeterministic computation)

Specifically: can we construct monads from graded monads?
Monads and graded monads

Given a monoid of grades:

$$(G, \cdot, 1)$$

(More generally, a monoidal category $(G, \cdot, 1)$.)

A $G$-graded monad consists of

- An endofunctor $T_g$ for each grade $g \in G$
  (with $T_g f : T_g X \to T_g Y$ for each $f : X \to Y$)
- A natural transformation $\eta_X : X \to T_1 X$
- A natural transformation $\mu_{g,g',X} : T_g (T_{g'} X) \to T_{g \cdot g'} X$ for each
  $g, g'$
  (satisfying unit and associativity laws)

Alternatively, have

$$f : X \to T_{g'} Y$$

$$\Rightarrow f : T_g X \to T_{g \cdot g'} Y$$
Monads and graded monads

Given a monoid of grades:

\[ (G, \cdot, 1) \]

(More generally, a monoidal category \((G, \cdot, 1)\).)

A **\(G\)-graded monad** consists of

- An endofunctor \(T_g\) for each grade \(g \in G\)
  (with \(T_g f : T_g X \to T_g Y\) for each \(f : X \to Y\))
- A natural transformation \(\eta_X : X \to T_1 X\)
- A natural transformation \(\mu_{g,g',X} : T_g(T_{g'}X) \to T_{g \cdot g'} X\) for each \(g, g'\)

(satisfying unit and associativity laws)

**Example (possibly-empty lists)**

- Grades are natural numbers with multiplication \((\mathbb{N}, \cdot, 1)\)
- Graded monad is:

\[
T_n X = \text{List}_{\leq n} X \quad \eta x = [x] \quad \mu xss = \text{concat} xss
\]
Monads and graded monads

Given a monoid of grades:

\((\mathcal{G}, \cdot, 1)\)

(More generally, a monoidal category \((\mathcal{G}, \cdot, 1)\).)

A \(\mathcal{G}\)-graded monad consists of

- An endofunctor \(T_g\) for each grade \(g \in \mathcal{G}\) (with \(T_g f : T_g X \to T_g Y\) for each \(f : X \to Y\))
- A natural transformation \(\eta_X : X \to T_1 X\)
- A natural transformation \(\mu_{g,g',X} : T_g(T_{g'}X) \to T_{g \cdot g'}X\) for each \(g, g'\) (satisfying unit and associativity laws)

Example (non-empty lists)

- Grades are positive integers with multiplication \((\mathbb{N}_+, \cdot, 1)\)
- Graded monad is:

\[
T_n X = \text{List}_{+\times n} X \quad \eta x = [x] \quad \mu \text{xs} = \text{concat xs}
\]
Monads from graded monads

*Can we turn graded monads $T$ into non-graded monads $\hat{T}$?*

For example:

- Can we construct a monad by constructing the corresponding graded monad first? (e.g. [Fritz and Perrone '18]'s Kantorovich monad)
- If we can model a language with grades, can we model the language without grades?

$$
\vdash \! g \ M : \text{int} \quad \longmapsto \quad [M] \in T_g \mathbb{Z}
$$

$$
\downarrow \quad \downarrow \lambda_g
$$

$$
\vdash \! M : \text{int} \quad \longmapsto \quad [M] \in \hat{T} \mathbb{Z}
$$

- Do we have

$$
\text{List}_{+\equiv} \mapsto \text{List}_+ \quad \text{List}_{-\equiv} \mapsto \text{List}
$$
Degradings

A degrading of a graded monad $(T, \eta, \mu)$ consists of

- A monad $(\hat{T}, \hat{\eta}, \hat{\mu})$
- Functions $\lambda_{g,X} : T_gX \to \hat{T}X$ preserving the structure, e.g. the multiplications:

$$
T_g(T_{g'}X) \xrightarrow{\mu} T_{g \cdot g'}X \\
\lambda_g \circ T_g \lambda_{g'} \quad \downarrow \quad \lambda_{g \cdot g'} \\
\hat{T}(\hat{T}X) \xrightarrow{\hat{\mu}} \hat{T}X
$$

Example: $(\text{List}_+, [-], \text{concat})$ forms a degrading of $(\text{List}_{+\equiv}, [-], \text{concat})$

$$
\lambda_{n,X} : \text{List}_{+\equiv n}X \subseteq \text{List}_+X
$$
Constructing degradings

Take the coproduct of \( g \mapsto T_g \):

\[
\hat{T} : \text{Set} \to \text{Set} \\
\hat{T}X = \sum_{g \in G} T_gX
\]

\[
\lambda_g : T_gX \to \hat{T}X \\
t \mapsto (g, t)
\]

so that elements of \( \hat{T}X \) are pairs \((g \in G, t \in T_gX)\)

- Have a unit

\[
\hat{\eta} : X \to \sum_{g \in G} T_gX \\
x \mapsto (1, \eta x)
\]

- But what about the multiplication?

\[
\hat{\mu} : \sum_{g \in G} T_g\left(\sum_{g' \in G} T_{g'}X\right) \rightarrow \sum_{g'' \in G} T_{g''}X
\]

from

\[
\mu_{g,g'} : T_g(T_{g'}X) \rightarrow T_{g.g'}X
\]
Algebraic coproducts

The coproduct $\hat{T}$ is an algebraic coproduct if:

- It forms a degrading

- For every other degrading $T'$, there are unique structure-preserving functions $\hat{T}X \to T'X$

  (more generally: algebraic Kan extension)

For models of effectful languages:

- A computation would be a pair of a $g$ and a computation of grade $g$

- For any other model given by a degrading $T'$, the unique functions preserve interpretations of terms
Algebraic coproducts

Algebraic Kan extensions sometimes exist:

Fritz and Perrone, A Criterion for Kan Extensions of Lax Monoidal Functors

but often don’t

- Neither List⁺≡ nor List⁻ has an algebraic coproduct
Algebraic coproducts

Algebraic Kan extensions sometimes exist:

Fritz and Perrone, A Criterion for Kan Extensions of Lax Monoidal Functors

but often don’t

▶ Neither $\text{List}_{+\equiv}$ nor $\text{List}_{\equiv}$ has an algebraic coproduct

Introduce two weakenings:

▶ Unique shallow degrading: don’t require structure-preservation for $\hat{T}X \to T’X$

▶ Initial degrading: don’t require a coproduct

Algebraic coproduct $\iff$ unique shallow degrading $\wedge$ initial degrading
First weakening: unique shallow degrading

If the coproduct $\hat{T}$ uniquely forms a degrading, call it the unique shallow degrading

- There are unique $\lambda$-preserving functions $\hat{T}X \to T'X$, but they don’t preserve all of the structure

Non-example

List does not form the unique shallow degrading of $\text{List}_-$

$$\hat{\mu} \text{ xss} = \text{concat} \text{ xss} \quad \text{or} \quad \hat{\mu} \text{ xss} = \begin{cases} [] & \text{if } [ ] \in \text{xss} \\ \text{concat} \text{ xss} & \text{otherwise} \end{cases}$$

Example

$(\text{List}_+, [-], \text{concat})$ is the unique shallow degrading of $\text{List}_{+-}$
List\(_+\) is a unique shallow degrading

If a non-empty list monad satisfies

\[
\mu \text{xss} = \text{concat xss} \quad \text{(for balanced xss)}
\]

then \(\mu = \text{concat}\)

Proof sketch:

1. Show that \(\mu \text{xss}\) cannot discard elements, by considering elements of \(\text{List}^3_+ X\)
2. Implies \(\mu\) cannot duplicate elements
3. Prove \(\mu[[x, y], [z]] = [x, y, z] = \mu[[x], [y, z]]\) by brute force
4. So \(\mu\) just concatenates, then permutes the result based on the length
5. These permutations must be identities
Second weakening: initial degrading

\(\hat{T}\) is the initial degrading of a graded monad \(T\) if:

- It is a degrading
- For any other degrading \(T'\), there are unique structure-preserving functions

\[\hat{T}X \rightarrow T'X\]

But: \(\hat{T}\) does not have to be the coproduct
(it is actually a Kan extension in \textbf{MonCat} instead of \textbf{Cat})
Constructing initial degradings

Start with a graded monad $T$

1. Take the (ordinary) coproduct of $g \mapsto Tg$
2. Construct the free monad on the coproduct
3. Quotient to get a degrading

These often exist, but are not intuitive:

- List$_\leq$ and List$_{\leq+}$ have initial degradings
- They don’t have simple descriptions: they are not List or List$_+$
Conclusions

There are:

- 2 monad structures on $\mathcal{P}$,
- a lot of monad structures on List.

Degradings are much more complicated than they first seem:

- List$_+$ is the unique shallow degrading, but not the initial degrading, of List$_+$
- List isn’t the unique shallow degrading or the initial degrading of List$_=$

Neither is an algebraic coproduct

See our PPDP’20 paper, and the Haskell code at

https://github.com/maciejpiorog/exotic-list-monads