List monads

Dylan McDermott Maciej Piróg Tarmo Uustalu

Outline

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

Outline

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

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:

$$\begin{array}{ccc} TX & \xrightarrow{\eta_{TX}} & T(TX) \\ {\scriptstyle T\eta_X}\downarrow & \searrow^{=} & \downarrow{\scriptstyle \mu_X} \\ T(TX) & \xrightarrow[\mu_X]{} & TX \end{array} \qquad \begin{array}{ccc} T(T(TX)) & \xrightarrow{\mu_{TX}} & T(TX) \\ {\scriptstyle T\mu_X}\downarrow & & \downarrow{\scriptstyle \mu_X} \\ T(TX) & \xrightarrow[\mu_X]{} & TX \end{array}$$

Example: the usual list monad is given by

$$\begin{array}{rcrcl} T = \mathrm{List} & : & \mathbf{Set} & \to & \mathbf{Set} \\ & & X & \mapsto & \text{set of finite possibly-empty lists over X} \\ & & f & \mapsto & \lambda[x_1, \ldots, x_n].\,[f x_1, \ldots, f x_n] \end{array}$$

$$\eta_X = \lambda x.\,[x] \qquad \mu_X = \lambda[\mathrm{xs}_1, \ldots, \mathrm{xs}_n].\ \mathrm{xs}_1 +\!\!+ \cdots +\!\!+ \mathrm{xs}_n$$

The usual list monad is given by

$$\begin{array}{rcl} T = \mathrm{List} \ : \ \mathbf{Set} & \to & \mathbf{Set} \\ X & \mapsto & \text{set of finite possibly-empty lists over X} \\ f & \mapsto & \lambda[x_1, \ldots, x_n].\, [f x_1, \ldots, f x_n] \end{array}$$

$$\eta_X = \lambda x.\, [x] \qquad \mu_X = \lambda[\mathrm{xs}_1, \ldots, \mathrm{xs}_n].\ \mathrm{xs}_1 +\!\!+ \cdots +\!\!+ \mathrm{xs}_n$$

Question: if the functor $T : \mathbf{Set} \to \mathbf{Set}$ is one of

List – *finite lists*
List_+ – *nonempty finite lists*
$\mathcal{P}$ – *subsets*
$\mathcal{P}_+$ – *nonempty subsets*
...

are there any monad structures other than the usual one?

All of the powerset monads

The covariant powerset functor $\mathcal{P} : \mathbf{Set} \longrightarrow \mathbf{Set}$, with

$$\mathcal{P} f = \lambda S. \{ f x \mid x \in S \} \qquad (f : X \longrightarrow Y)$$

forms a monad in exactly two ways.

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

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

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

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

The nonempty powerset functor $\mathcal{P}_+ : \mathbf{Set} \longrightarrow \mathbf{Set}$ 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

$$\begin{array}{ccc} \mathcal{P}S & \xrightarrow{\alpha_S} & \mathcal{P}S \\ {\scriptstyle \mathcal{P}\subseteq}\downarrow & & \downarrow{\scriptstyle \mathcal{P}\subseteq} \\ \mathcal{P}X & \xrightarrow[\alpha_X]{} & \mathcal{P}X \end{array}$$

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 $y \in S$,

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

so $y \in \mathcal{P}\mathrm{swap}_{x,y}(\alpha_X S) = \alpha_X S$.

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

$$\begin{CD} \mathcal{P}X @>{\alpha_X}>> \mathcal{P}X \\ @V{\mathcal{P}\langle\rangle}VV @VV{\mathcal{P}\langle\rangle}V \\ \mathcal{P}1 @>>{\alpha_1}> \mathcal{P}1 \end{CD}$$

So α is one of

$$\alpha_X = \lambda S.\,\emptyset \qquad\qquad \alpha_X = \lambda S.\,S$$

All of the powerset monads

By similar proofs, the unit and multiplication

$$\eta_X : X \to \mathcal{P}X \qquad \mu_X : \mathcal{P}(\mathcal{P}X) \to \mathcal{P}X$$

are completely determined by

$$\eta_1 : 1 \to \mathcal{P}1 \qquad \mu_1 : \mathcal{P}(\mathcal{P}1) \to \mathcal{P}1$$

but only two pairs (η_1, μ_1) satisfy the monad laws.

Lists are harder

Natural transformations

$$\alpha_X : \mathrm{List}X \longrightarrow \mathrm{List}X$$

are not completely determined by

$$\alpha_1 : \mathrm{List}1 \longrightarrow \mathrm{List}1$$

For example

$$\mathrm{id}_1 = \mathrm{reverse}_1 \quad \text{but} \quad \mathrm{id} \neq \mathrm{reverse}$$

Lists are harder

Natural transformations

$$\alpha_X : \mathrm{List}X \longrightarrow \mathrm{List}X$$

are not completely determined by

$$\alpha_1 : \mathrm{List}1 \longrightarrow \mathrm{List}1$$

They *are* completely determined by

$$\alpha_2 : \mathrm{List}2 \longrightarrow \mathrm{List}2$$

but this doesn't seem to help much.

What we can say for certain

For $T = \mathrm{List}$:

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

$$(\lambda(\mathrm{xs}_1, \ldots, \mathrm{xs}_n).\ \mu_X [\mathrm{xs}_1, \ldots, \mathrm{xs}_n]) : (\mathrm{List}\, X)^n \to \mathrm{List}\, X$$

Similar things hold for $T = \mathrm{List}_+$.

Some possibly-empty list monads

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

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

$$\begin{aligned} \varepsilon \cdot x \;&=\; x \;=\; x \cdot \varepsilon \\ (x \cdot y) \cdot z \;&=\; x \cdot (y \cdot z) \end{aligned} \qquad\qquad \begin{aligned} \varepsilon \cdot x \;&=\; \varepsilon \;=\; x \cdot \varepsilon \\ (x \cdot y) \cdot z \;&=\; x \cdot (y \cdot z) \end{aligned}$$

Some possibly-empty list monads

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

$$\begin{aligned} \varepsilon \cdot x &= \varepsilon = x \cdot \varepsilon \\ (x \cdot y) \cdot z &= x \cdot (y \cdot (x \cdot z)) \end{aligned}$$

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

Some possibly-empty list monads

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 = { [] concat xss	if exists null xss otherwise
$\varepsilon \cdot x = \varepsilon$ $x \cdot \varepsilon = \varepsilon$ $(x \cdot y) \cdot z = x \cdot (y \cdot (x \cdot z))$	[] concat (map palindromise (init xss)) ++ last xss	if null xss or exists null xss otherwise
$\varepsilon \cdot x = \varepsilon$ $x \cdot \varepsilon = \varepsilon$ $(x \cdot y) \cdot z = \varepsilon$	[] [] map head (init xss) ++ last xss	if null xss or exists null xss else if exists (not ∘ sglt) (init xss) otherwise
$\varepsilon \cdot x = x$ $x \cdot \varepsilon = \varepsilon$ $(x \cdot y) \cdot z = y \cdot z$	[] [] concat (map safeLast (init xss)) ++ last xss	if null xss else if null (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** $\cdot$

Some possibly-empty list monads

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

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

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

$$(\lambda(\mathrm{xs}_1, \ldots, \mathrm{xs}_n).\ \mu_X[\mathrm{xs}_1, \ldots, \mathrm{xs}_n]) : (\mathrm{List}\,X)^n \to \mathrm{List}\,X \qquad (n \le p)$$

generate lists of length $\le p$.

How many list monads are there?

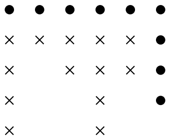
Answer: infinitely many

- Can discard elements
- Can duplicate elements
- Can have no finite presentation

Some non-empty list monads

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

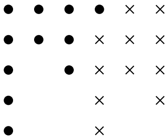
$$\mu\,[\mathrm{xs}_1, \ldots, \mathrm{xs}_n] \;\; = \;\; \mathrm{head}\,\mathrm{xs}_1 :: \cdots :: \mathrm{head}\,\mathrm{xs}_{n-1} :: \mathrm{xs}_n$$



Some non-empty list monads

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

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

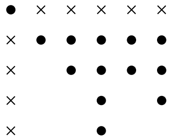


Requires infinitely many operators!

Some non-empty list monads

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

$$\mu\,\mathrm{xss} \;=\; \mathrm{head}\,(\mathrm{head}\,\mathrm{xss}) :: \mathrm{concat}(\mathrm{tail}\,(\mathrm{List}_+\,\mathrm{tail}\,\mathrm{xss}))$$



This arises from $\mathrm{List}_+ \cong \mathrm{Id} \times \mathrm{List}$

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 = lists over X)
- Graded monads organize computations into sets T_gX
  (e.g. T_gX = lists over X 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 = lists over X)
- Graded monads organize computations into sets T_gX (e.g. T_gX = lists over X 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:

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

(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)

Alternatively, have

$$\frac{f : X \to T_{g'} Y}{\gg\!= f : T_g X \to T_{g \cdot g'} Y}$$

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 (possibly-empty lists)

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

$$T_n X = \mathrm{List}_{=n} X \qquad \eta\, x = [x] \qquad \mu\, \mathrm{xss} = \mathrm{concat}\, \mathrm{xss}$$

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:

$$T_n X = \mathrm{List}_{+=n} X \qquad \eta\, x = [x] \qquad \mu\, \mathrm{xss} = \mathrm{concat}\, \mathrm{xss}$$

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?

$$\begin{array}{ccc} \vdash_g M : \mathbf{int} & \longmapsto & [\![M]\!] \in T_g\,\mathbb{Z} \\ \downarrow & & \downarrow \lambda_g \\ \vdash \underline{M} : \mathbf{int} & \longmapsto & [\![\underline{M}]\!] \in \hat{T}\,\mathbb{Z} \end{array}$$

- Do we have

$$\mathrm{List}_{+=} \mapsto \mathrm{List}_{+} \qquad \mathrm{List}_{=} \mapsto \mathrm{List}$$

Degradings

A degrading of a graded monad (T, η, μ) consists of

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

$$\begin{array}{ccc} T_g(T_{g'}X) & \xrightarrow{\mu} & T_{g \cdot g'}X \\ {\scriptstyle \lambda_g \circ T_g \lambda_{g'}} \big\downarrow & & \big\downarrow {\scriptstyle \lambda_{g \cdot g'}} \\ \hat{T}(\hat{T}X) & \xrightarrow[\hat{\mu}]{} & \hat{T}X \end{array}$$

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

$$\lambda_{n,X} : \mathrm{List}_{+=n} X \subseteq \mathrm{List}_{+} X$$

Constructing degradings

Take the coproduct of $g \mapsto T_g$:

$$\hat{T} : \mathbf{Set} \longrightarrow \mathbf{Set} \qquad \lambda_g : T_g X \longrightarrow \hat{T} X$$
$$\hat{T}\, X = \textstyle\sum_{g \in \mathcal{G}} T_g X \qquad t \mapsto (g, t)$$

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

- Have a unit

$$\hat{\eta} : X \longrightarrow \textstyle\sum_{g \in \mathcal{G}} T_g X$$
$$x \mapsto (1, \eta\, x)$$

- But what about the multiplication?

$$\hat{\mu} : \textstyle\sum_{g \in \mathcal{G}} T_g \big(\sum_{g' \in \mathcal{G}} T_{g'} X \big) \xrightarrow{?} \sum_{g'' \in \mathcal{G}} T_{g''} X$$

from

$$\mu_{g,g'} : T_g(T_{g'} X) \longrightarrow T_{g \cdot g'} X$$

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 $\mathrm{List}_{+=}$ nor $\mathrm{List}_{=}$ 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 $\mathrm{List}_{+=}$ nor $\mathrm{List}_{=}$ 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 $\Leftrightarrow$ 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 λ-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 $\mathrm{List}_{=}$

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

Example

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

List_+ is a unique shallow degrading

If a non-empty list monad satisfies

$$\mu\,\mathrm{xss} = \mathrm{concat}\,\mathrm{xss} \qquad \text{(for balanced xss)}$$

then $\mu = \mathrm{concat}$

Proof sketch:

1. Show that $\mu\,\mathrm{xss}$ cannot discard elements, by considering elements of $\mathrm{List}_+^3 X$
2. Implies μ cannot duplicate elements
3. Prove $\mu[[x, y], [z]] = [x, y, z] = \mu[[x], [y, z]]$ by brute force
4. So μ 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

$$\hat{T}X \to T'X$$

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

Constructing initial degradings

Start with a graded monad T

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

These often exist, but are not intuitive:

- $\mathrm{List}_=$ and $\mathrm{List}_{+=}$ 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 $\mathrm{List}_{+=}$
- List isn't the unique shallow degrading or the initial degrading of $\mathrm{List}_{=}$

Neither is an algebraic coproduct

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

`https://github.com/maciejpirog/exotic-list-monads`