List Monads

Dylan McDermott Maciej Piróg Tarmo Uustalu

Theory Days in Riga, 6–8 May 2022

Monads and the list monad in Haskell

```
class Functor f where
  fmap :: (a -> b) -> f a -> f b

instance Functor [] where              -- [] is the list type
  fmap = map

{-
  fmap id == id
  fmap (g . f) == fmap g . fmap f
-}

class Functor t => Monad t where
  unit :: a -> t a
  mult :: t (t a) -> t a

instance Monad [] where
  unit x = [x]
  mult xss = concat xss

{-
  mult . unit == id
  mult . fmap unit == id
  mult . mult == mult . fmap unit
-}
```

List monads?!

- In semantics, we tend to talk about the list monad (or the nonempty list monad), on Set or some other category.

- We mean the free monoid monad (or the free semigroup monad).
- But this is not the only monad structure on the list functor (resp. the nonempty list functor).

- There are many other list monads.
- In fact, infinitely many, and many of them strange and unexpected.
- We do not know today how to describe them all.

The free monoid monad

- The *free monoid monad* is $(\mathrm{List}, \eta, \mu)$ where

  $\eta\,\mathrm{x} = [\mathrm{x}]$
  $\mu\,[\mathrm{xs}_1, \ldots, \mathrm{xs}_n] = \mathrm{concat}\ [\mathrm{xs}_1, \ldots, \mathrm{xs}_n] = \mathrm{xs}_1 +\!\!+ \ldots +\!\!+ \mathrm{xs}_n$

- It is presented by a nullary operation ε, a binary operation $\cdot$ and equations

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

- This means that lists over X are isomorphic to terms formed of these operations with variables from X, identified up to these equations.
- Moreover, the singleton function and concatenation agree with variables-as-terms and terms-of-terms as terms.

The free semigroups-with-zero monad

- The free monoid monad is not the only monad structure on List.
- There is, e.g., the free *semigroups-with-zero* monad $(\mathrm{List}, \eta, \mu)$ where

  $\eta\,\mathrm{x} = [\mathrm{x}]$

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

- This monad is presented by a nullary operation ε, a binary operation $\cdot$ and equations

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

Hmmm...

- Are the further monad structures on functors List or List_+?
- What are they like?
- How to construct such monads systematically?
- Can we describe them all in some useful manner?

Some easy observations

- The following must hold about any monad structure on List or List_+.
- $\eta\,\mathrm{x} = \mathrm{replicate}\,e\,x = [\underbrace{\mathrm{x},\ldots,\mathrm{x}}_{e\ \text{times}}]$ for some $e \geq 1$ not depending on x.
- μ cannot invent elements. If x appears in $\mu\,\mathrm{xss}$, then it must appear in xss.
- The monad has a presentation with possibly infinitely many operations of finite arity.
- When $e = 1$, a set of operations denoting some of these will always do:

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

  ($n \geq 0$ for List, $n \geq 1$ for List_+.)

Monads $(\mathrm{List}, [-], \mu)$ presented with ε, $\cdot$

- Consider monad structures on List with $\eta\,\mathrm{x} = [\mathrm{x}]$.

- There are infinitely many monad structures presented with one nullary operation ε, one binary operator $\cdot$ denoting
  $[\![\varepsilon]\!] = \mathrm{list}_0^\sharp = \mu\,[]$
  $\mathrm{xs}\;[\![\cdot]\!]\;\mathrm{ys} = \mathrm{list}_2^\sharp\,(\mathrm{xs}, \mathrm{ys}) = \mu\,[\mathrm{xs}, \mathrm{ys}]$

- There are infinitely many such monad structures already when the equations take the very specific form

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

  resembling the equations of a monoid.

Monads $(\mathrm{List}, [-], \mu)$ presented with ε, $\cdot$

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
$\varepsilon \cdot x = \varepsilon$ $x \cdot \varepsilon = x^{n+2}$ $(x \cdot y) \cdot z = x \cdot y$	[] if null xss replicateLast $(n+1)$ else if null (head (dropWhile sglt (init xss) (map head (takeWhile sglt (init xss))) ++[last xss])) map head (takeWhile sglt (init xss)) otherwise ++ head (dropWhile sglt (init xss) ++ [last xss])
$\varepsilon \cdot x = \varepsilon$ $x \cdot \varepsilon = x^{n+2}$ $(x \cdot y) \cdot z = x^{m+2}$	[] if null xss replicateLast $(n+1)$ else if null (head (dropWhile sglt (init xss) (map head (takeWhile sglt (init xss))) ++[last xss])) map head (takeWhile sglt (init xss)) else if exists (not ∘ sglt) (init xss) ++ replicate $(m+2)$ or null (last xss) (head (head (dropWhile sglt (init xss)))) map head (init xss) ++ last xss otherwise
$\varepsilon \cdot x = \varepsilon$ $x \cdot \varepsilon = x^{n+2}$ $(x \cdot y) \cdot z = \varepsilon$	[] if null xss replicateLast $(n+1)$ else if exists (not ∘ sglt) (init xss) (map head (takeWhile sglt (init xss))) or null (last xss) map head (init xss) ++ last xss otherwise

Palindromes

- Here is just one example of another such monad beyond the free monoids monad and the free semigroups-with-zero monad.

- The equations

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

yield a monad structure on List with

$$\mu\,\mathrm{xss} = \begin{cases} [] & \text{if } \mathrm{null}\,\mathrm{xss} \\ [] & \text{if } \mathrm{exists}\,\mathrm{null}\,\mathrm{xss} \\ \mathrm{concat}\ (\mathrm{map}\,\mathrm{palindromise}\,(\mathrm{init}\,\mathrm{xss})) +\!\!+ \mathrm{last}\,\mathrm{xss} & \text{otherwise} \end{cases}$$

where

$$\mathrm{palindromise}\,\mathrm{xs} = \mathrm{xs} +\!\!+ \mathrm{reverse}\,(\mathrm{init}\,\mathit{xs})$$

A generic (candidate) multiplication

- How did we find these multiplications?
- Given some *rhsL*, *rhsR*, *rhsA*, they may or may not yield a weakly normalizing and confluent TRS that has lists a representation of its unique normal forms.
- This definition of a candidate μ just assumes this to be the case.

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

- We can quickcheck if this candidate μ verifies the monad equations.
- If it does, it is worth trying to prove manually that the TRS is indeed good.

A monad with no finite presentation

- This is a monad structure on List with no finite presentation.

  $\eta\,\mathrm{x} = [\mathrm{x}]$

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

- We know that any presentation of a monad with this η must be reducible one with some of the operations $\mathrm{list}^\sharp_n$.
- For this monad, however we choose k, with operations $\mathrm{list}^\sharp_n$, $n \leq k$, one can only produce lists of length at most k.

Monads $(\mathrm{List}_+, [-], \mu)$ presented with $\cdot$

- There are also infinitely many monad structures on List_+ with $\eta\,[\mathrm{x}] = \mathrm{x}$ and one binary operation $\cdot$ interpreting into

  $\mathrm{xs}\ [\![\cdot]\!]\ \mathrm{ys} = \mathrm{list}_2^\sharp\,(\mathrm{xs}, \mathrm{ys}) = \mu\,[\mathrm{xs}, \mathrm{ys}]$

  and have just one equation of the form

$$(x \cdot y) \cdot z = \mathit{rhsA}$$

Monads $(\mathrm{List}_+, [-], \mu)$ presented with $\cdot$

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

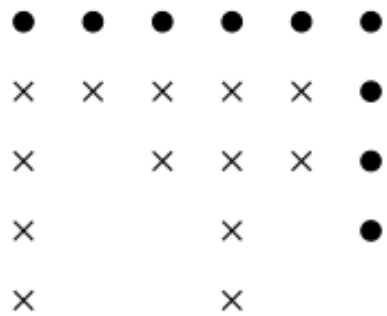
Hybrid monad of Neves

- The equation

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

  gives a nonempty list monad with this multiplication:

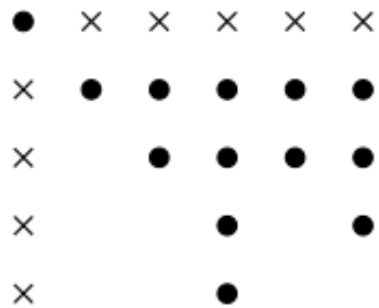
  $\mu\,\mathrm{xss} = \mathrm{map}\,\mathrm{head}\ (\mathrm{init}\,\mathrm{xss}) +\!\!+ \mathrm{last}\,\mathrm{xss}$



A monad structure on List_+ with η not $[-]$

- Here is a monad structure on List_+ with a different unit.

  $\eta\,\mathrm{x} = [\mathrm{x}, \mathrm{x}]$
  $\mu\,(\mathrm{xs} :: \mathrm{xss}) = \mathrm{head}\ \mathrm{xs} :: \mathrm{concat}\ (\mathrm{map}\,\mathrm{tail}\ \mathrm{xss})$



- This monad arises as the product of Id and $(\mathrm{List}, [-], \mathrm{concat}\,)$.
- In the same way, any monad structure on List with unit $\eta\,\mathrm{x} = [\mathrm{x}]$ yields a monad structure on List_+ with unit $\eta\,\mathrm{x} = [\mathrm{x}, \mathrm{x}]$.

A non-product monad structure on List_+ with $\eta\,\mathrm{x} = [\mathrm{x}, \mathrm{x}]$

- Here is a different multiplication for $\eta\,\mathrm{x} = [\mathrm{x}, \mathrm{x}]$:

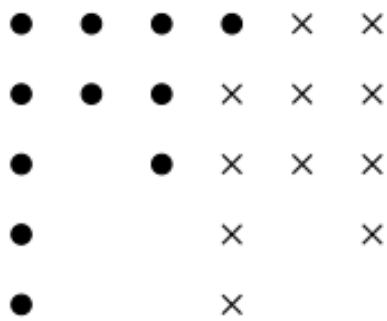
$$\mu\,\mathrm{xss} = \begin{cases} [\mathrm{head}\ (\mathrm{head}\ \mathrm{xss})] & \text{if } \mathrm{sglt}\,\mathrm{xss} \text{ or } \mathrm{sglt}\,(\mathrm{last}\,\mathrm{xss}) \\ \mathrm{map}\,\mathrm{head}\ (\mathrm{init}\,\mathrm{xss}) & \text{otherwise} \\ \quad +\!\!+\mathrm{tail}\ (\mathrm{last}\,\mathrm{xss})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

- This monad is <u>not</u> a product of Id and a List monad.

$(\mathrm{List}_+, [-], \mathrm{concat})$ is not identifiable by finite testing

- For any $p \geq 2$, it is the case that $(\mathrm{List}_+, [-], \mathrm{concat}_p)$ is a monad where

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



- We have $\mathrm{concat}_p\,\mathrm{xss} = \mathrm{concat}\ \mathrm{xss}$ for any xss such that $|\mathrm{concat}\ \mathrm{xss}| \leq p!$

- This monad has no finite presentation.
- With operations $\mathrm{list}^\sharp_n$, $n \leq k$, one can only produce nonempty lists of length at most $\max(k, p)$.

Variant monads via $\mathrm{reverse}$

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

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

- (Note that $\mathrm{concat}^R = \mathrm{concat}$.)

- The same applies to List_+.

Some simple to state open questions

- We have monads on List and List_+ where μ permutes elements.

  Are there μ that can permute elements without duplicating or deleting any?

- We know monad structure on List_+ with $\eta = \mathrm{replicate}\,2$.

  Are there monads on List with $\eta = \mathrm{replicate}\,2$ or monads on List_+ with $\eta = \mathrm{replicate}\,3$?

- The monad $(\mathrm{List}, [-], \mathrm{concat}')$ arises thanks to $\mathrm{List} = \mathrm{Maybe} \cdot \mathrm{List}_+$ from a distributive law of $(\mathrm{List}_+, [-], \mathrm{concat})$ over Maybe.

  Do some other monads on List_+ distribute over Maybe?

List monads with $\mu = \mathrm{concat}$ on balanced input

- Say that a list of lists is *balanced* if its inner lists are of equal length.

- There are at least 2 monad structures on List with $\mu = \mathrm{concat}$ on balanced input:

  $\mu = \mathrm{concat}$ and also $\mu = \mathrm{concat}'$.

- There is exactly 1 monad structure on List_+ with $\mu = \mathrm{concat}$ on balanced input:

  only $\mu = \mathrm{concat}$ works.

- The proof of this fact is difficult and involves a number of steps.

Proof outline

- Assume $\mu = \mathrm{concat}$ on balanced input. We then show the following.
- μ does not delete elements.
- μ does not duplicate elements.
- Since μ cannot invent new elements, $\mu\,\mathrm{xss}$ is therefore a permutation of $\mathrm{concat}\,\mathrm{xss}$.
- $\mu\,[[\mathrm{x},\mathrm{y}],[\mathrm{z}]] = [\mathrm{x},\mathrm{y},\mathrm{z}] = \mu\,[[\mathrm{x}],[\mathrm{y},\mathrm{z}]]$: machine check of all possible partial μ defined on input of total length ≤ 6; there are 120 such functions.
- 7 very specific lemmata.
- By induction on total length of xss, one gets $\mu\,\mathrm{xss} = \mathrm{concat}\ \mathrm{xss}$.

Takeaway

- There are very many list monads, of different flavors, some very bizarre.
- There are exactly 2 monad structures on $\mathcal{P}$.
- There are infinitely many already on $\mathcal{M}$, but the situation seems considerably simpler than in the case of List.
- We do not know how to even approach the task of describing all monads on List.
- These are cool combinatorial problems.
- Cf. the work on distributive laws between the (standard) powerset, multiset and list monads by Klin and Salamanca, Zwart.
  This is similar in spirit.

Paper and code

- Check our paper:

  D. McDermott, M. Piróg, T. Uustalu. Degrading lists.
  Proc. of PPDP '20, ACM, 2020.

- Have fun with our Haskell library of exotic list monads:

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