NOTES ON LAX MONOIDAL FUNCTORS, MONADS, AND HASKELL

IORDAN GANEV

Version 2.1

Contents

1. Introduction

These notes are an exploration of lax monoidal functors and monads in category theory as they relate to applicatives and monads in Haskell. Of particular interest are the categorical semantics behind the `sequence` map in Haskell. While our focus is on mathematical foundations, we also have practical goals of understanding how best to use the `Monad` and `Applicative` type classes. We assume familiarity with the basics of category theory, and exposure to Haskell. Recommended resources include [oH, Rie17, Mil19].

1.1. Monads in Haskell.

Haskell is a strongly typed language, and contains various ways to produce new types based on old types; these are known as *type constructors*. Roughly speaking, a monad `m` is a type constructor with desirable properties with respect to composition. The monad properties allow for a consistent and associative way to "compose" a function of type `a -> m b` with one of type `b -> m c` to produce one of type `a -> m c`:

```
(a -> m b) -> (b -> m c) -> a -> m c
```

Monads also include a unit function `a -> m a`. In many cases, it is helpful to regard values of type `m a` as operations that can be executed to produce a value of type `a`. So the

composition rule can be interpreted giving a consistent way to combine two executions into one. We discuss monads in Haskell in more detail in Section 5. Examples include:

- The maybe type constructor `Maybe a = Nothing | Just a`.
- The list type constructor `[a] = Nil | Cons a [a]`.
- The state type constructor `State s a = State (s -> (a,s))`, where `s` is the state and `a` is the argument of the monad.
- The representable type constructor `Repr e a = Repr (e -> a)` wher `e` is considered fixed and `a` is the argument of the monad.

A weaker version of monads are *applicatives*; and applicative `f` is a type constructor, together with a unit function `a -> f a`, and either of the following equivalent operations:

```
<*> ::  f (a -> b) -> f a -> f b
liftA2 ::  (a -> b -> c) -> f a -> f b -> f c
>*< ::  f a -> f b -> f (a, b)
```

The first provides a way of "distributing" the functor into the domain and codomain of a function; the second lifts a map on a product to a map where the functor is applied to each factor as well as to the codomain; the third "factors out" the functor from a product. Every applicative is a monad, while examples of applicatives that are not monads include:

- The validation applicative `Val a = Ok a | Error s` for a monoid `s` (for example, the list monoid `s=[b]`).
- The ziplist type constructor which differs from ordinary lists in that (1) infinite lists are allowed and (2) the unit picks out constant infinite lists.

We discuss these and other examples in detail in Sections 5 and 6.

1.2. **Monads in category theory.** In category theory, a *monad* on a category $\mathcal{C}$ is an endofunctor $T : \mathcal{C} \to \mathcal{C}$ with properties that allow for a consistent and associative composition rule:

$$\mathcal{C}(X, TY) \times \mathcal{C}(Y, TZ) \to \mathcal{C}(X, TZ)$$

where we write $\mathcal{C}(-,-)$ for the set of morphisms from one object to another. This composition rule is part of the definition of the *Kleisli category* associated to the monad T, which we discuss further in Section 4. Moreover, the composition rule is equivalent to requiring that T be a monoid in the category of endofunctors (with the operation of composition), which is the usual method of introducing monads in category theory. In Section 6, we discuss the following examples of monads on the category of sets:

(1) The maybe monad $TX = 1 + X$.
(2) The list monad $TX = [X] = \sum_{n \in \mathbb{N}} X^n$.
(3) The state monad $TX = \mathrm{Hom}(S, X \times S)$ for a fixed set S.
(4) The free module monad $TX = \mathrm{Hom}_{\mathrm{f}}(X, R)$ where R is a semiring (such as the natural numbers $\mathbb{N}$ or the real numbers $\mathbb{R}$), and the subscript "f" denotes finitely supported functions.
(5) The representable monad $TX = \mathrm{Hom}(E, X)$ for a fixed set E.

Meanwhile, an applicative in Haskell is parallel to a *lax monoidal functor* on the category of sets, which refers to an endofunctor A on the category of sets with a natural transformation

$$AX \times AY \to A(X \times Y)$$

that allows us to "factor out" the functor from products. We define lax monoidal functors precisely in Section 3 and show that every monad is a lax monoidal functor in Section 4. The converse is false, as illustrated by examples in Section 6.

2. Preliminaries

2.1. **Notation.** For a category $\mathcal{C}$, we write $\mathcal{C}(X, Y)$ for the set of morphisms from the object X to the object Y. Mostly, we work with the category Set of sets; we write $\mathrm{Hom}(X, Y) = \mathsf{Set}(X, Y)$ for the set of morphisms from a set X to a set Y. Recall the product-Hom adjunction:

$$\mathrm{Hom}(X, \mathrm{Hom}(Y, Z)) \simeq \mathrm{Hom}(X \times Y, Z) \simeq \mathrm{Hom}(Y, \mathrm{Hom}(X, Z))$$

for any sets X, Y, and Z. Additionally, we write $X + Y$ for the coproduct (disjoint union) of two sets X and Y, which are referred to as the cofactors of the coporduct. Given maps $f : X \to Z$ and $g : Y \to Z$, we write $\langle f, g \rangle$ for the induced map $X + Y \to Z$. We also write $\sum_{i \in I} X_i$ for the coproduct of an indexed family of sets.

Our use of the symbol '1' is overloaded; its meaning will be clear from context to be either the identity functor, the identity map, the one-point set, the single element of the one-point set, the unit of a semiring, or the number one.

2.2. **Endofunctors.** For any endofunctor F of Set, there is a map

$$\mathrm{Hom}(X \times Y, Z) \longrightarrow \mathrm{Hom}(FX \times FY, F^2 Z) \tag{2.1}$$

natural in X, Y, and Z, given as follows:

$$\begin{aligned}
\mathrm{Hom}(X \times Y, Z) &\xrightarrow{\sim} \mathrm{Hom}(X, \mathrm{Hom}(Y, Z)) \xrightarrow{F \circ -} \mathrm{Hom}(X, \mathrm{Hom}(FY, FZ)) \\
&\xrightarrow{\sim} \mathrm{Hom}(X \times FY, FZ) \\
&\xrightarrow{\sim} \mathrm{Hom}(FY, \mathrm{Hom}(X, FZ)) \xrightarrow{F \circ -} \mathrm{Hom}(FY, \mathrm{Hom}(FX, F^2 Z)) \\
&\xrightarrow{\sim} \mathrm{Hom}(FX \times FY, F^2 Z)
\end{aligned}$$

The first, third, fourth, and sixth maps use the product-Hom adjunction. The second and fifth are post-compositions with the map on morphisms provided by the functor F. As a formula, $f : X \times Y \to Z$ is sent to the function

$$\begin{aligned}
FX \times FY &\to F^2 Z \\
(\tilde{x}, \tilde{y}) &\mapsto F[x \mapsto F[y \mapsto f(x, y)](\tilde{y})](\tilde{x})
\end{aligned}$$

As a special case, we have a distinguished morphism $FX \times FY \to F^2(X \times Y)$ arising as the image of the identity morphism on $X \times Y$.

Example 2.1. Let E be a fixed set, and consider the representable endofunctor $\mathrm{Hom}(E,-)$ on the category Set. The map in Equation 2.1 takes $f : X \times Y \to Z$ to the map

$$\begin{aligned}\mathrm{Hom}(E,X) \times \mathrm{Hom}(E,Y) &\to \mathrm{Hom}(E, \mathrm{Hom}(E,Z)) \\ (\phi,\psi) &\mapsto [e \mapsto [e \mapsto f(\phi(e),\psi(e))]]\end{aligned}$$

2.3. **The list endofunctor.** Let $[X] = \mathsf{List}(X) = \sum_{n=0}^{\infty} X^n$ be the set of all (finite) lists with entries in the set X; this defines an endofunctor $[-] : \mathsf{Set} \to \mathsf{Set}$, where a map $f : X \to Y$ is sent to the map $[x_1,\ldots,x_n] \mapsto [f(x_1),\ldots,f(x_n)]$. Given a list in $[X]$ and an element in X, we can form a new list by attaching the element to the front of the list. This is traditionally known as the cons map:

$$\mathsf{cons}_X : X \times [X] \to [X]; \qquad (x,[x_1,\ldots,x_n]) \mapsto [x,x_1,\ldots,x_n]$$

The image of cons_X comprises all non-empty lists. Including the empty list via the map $\mathsf{nil}_X : 1 \to [X]$ gives an isomorphism:

$$\langle \mathsf{nil}_X, \mathsf{cons}_X \rangle : 1 + X \times [X] \xrightarrow{\sim} [X]$$

where we apply nil_X to the first cofactor and cons_X to the second.

Example 2.2. The map of Equation 2.1 specialized to the list endofunctor $F = [-]$ takes $f : X \times Y \to Z$ to the map $[X] \times [Y] \to [[Z]]$ defined as:

$$\begin{aligned}(\mathbf{x},\mathbf{y}) \mapsto [&[f(x_1,y_1), f(x_1,y_2), \ldots, f(x_1,y_m)] \\ &[f(x_2,y_1), f(x_2,y_2), \ldots, f(x_2,y_m)] \\ &\vdots \\ &[f(x_n,y_1), f(x_3,y_2), \ldots, f(x_n,y_m)]]\end{aligned}$$

where $\mathbf{x} = [x_1,x_2,\ldots,x_n]$ and $\mathbf{y} = [y_1,y_2,\ldots,y_m]$. In particular, the output is a list of $n = \mathrm{len}(x)$ lists, each of length $m = \mathrm{len}(y)$. If $\mathbf{x} = []$, then the output is the empty list, while if $\mathbf{y} = []$, then the output is a list of $n = \mathrm{len}(\mathbf{x})$ empty lists.

3. Lax monoidal functors

Unless specified otherwise, all endofunctors are of the category Set of sets.

3.1. **Definitions.** We begin directly with the definition of a lax monoidal endofunctor.

Definition 3.1. A lax monoidal endofunctor is an endofunctor $A : \mathsf{Set} \to \mathsf{Set}$ together with a morphism $\psi : 1 \to A1$, where 1 is the one-point set, and a natural transformation

$$\alpha_{X,Y} : AX \times AY \to A(X \times Y)$$

that satisfy the commutative diagrams:

$$\begin{array}{ccc} AX \times AY \times AZ & \xrightarrow{\alpha_{X,Y} \times \mathrm{id}} & A(X \times Y) \times AZ \\ \downarrow{\scriptstyle \mathrm{id} \times \alpha_{Y,Z}} & & \downarrow{\scriptstyle \alpha_{X \times Y,Z}} \\ AX \times A(Y \times Z) & \xrightarrow{\alpha_{X,Y \times Z}} & A(X \times Y \times Z) \end{array}$$

$$\begin{array}{ccc} 1 \times AX & \xrightarrow{\sim} & AX \\ {\scriptstyle \psi \times \mathrm{id}}\downarrow & & \uparrow{\scriptstyle \sim} \\ A1 \times AX & \xrightarrow[\alpha_{1,X}]{} & A(1 \times X) \end{array} \qquad \begin{array}{ccc} AX \times 1 & \xrightarrow{\sim} & AX \\ {\scriptstyle \mathrm{id} \times \psi}\downarrow & & \uparrow{\scriptstyle \sim} \\ AX \times A1 & \xrightarrow[\alpha_{X,1}]{} & A(X \times 1) \end{array}$$

These are known as the associativity, left unit, and right unit axioms, respectively.

Remark 3.2. A functor is monoidal if α is a natural isomorphism and ψ is an isomorphism; one can regard lax monoidal is a weak version of monoidal. Lax monoidal functors are monoids in the category of endofunctor of Set with the Day convolution, see [Mil17] for more details. Lax monoidal functors are the category-theoretic version of applicatives in Haskell, hence our choice of the symbol A.

Example 3.3. The representable endofunctor $\mathrm{Hom}(E, -)$ is lax monoidal, and in fact monoidal. The unit map $1 \to \mathrm{Hom}(E, 1)$ is an isomorphism as there is a single map $E \to 1$. The natural transformation α takes a pair of maps $\phi : E \to X$ and $\psi : E \to Y$ to the map $E \to X \times Y$ defined as $e \mapsto (\phi(e), \psi(e))$; this is also an isomorphism. The axioms are easily verified.

We postpone a detailed discussion of further examples to Section 6.

3.2. **The sequence map.** One upshot of lax monoidal functors is the existence of the so-called "sequence map", which is a commutation relation between any lax monoidal endofunctor and the list endofunctor. The definition uses the nil and cons maps from Section 2.3.

Definition 3.4. Let (A, ψ, α) be a lax monoidal functor. Define a natural transformation (natural in X):

$$\begin{aligned} \mathsf{seq}_X : [AX] &\to A[X] \\ \mathsf{nil}_{AX}(1) &\mapsto A(\mathsf{nil}_X) \circ \phi(1) \\ \mathsf{cons}_{AX}(p, \mathbf{p}) &\mapsto A(\mathsf{cons}_X) \circ \alpha_{X,[X]}(p, \mathsf{seq}_X(\mathbf{p})) \end{aligned}$$

The definition uses the fact that $\langle \mathsf{nil}_{AX}, \mathsf{cons}_{AX} \rangle$ defines an isomorphism $1 + AX \times [AX] \to [AX]$. We have the following commutative diagram:

$$\begin{array}{ccccc} 1 + AX \times [AX] & & \xrightarrow{\quad \langle\ \mathsf{nil}_{AX}\ ,\ \mathsf{cons}_{AX}\ \rangle \quad} & & [AX] \\ {\scriptstyle \langle 1, 1 \times \mathsf{seq}_X \rangle}\downarrow & & & & \downarrow{\scriptstyle \mathsf{seq}_X} \\ 1 + AX \times A[X] & \xrightarrow{\ \psi\ +\ \alpha_{X,[X]}\ } & A1 + A(X \times [X]) & \xrightarrow{\ \langle\ A(\mathsf{nil}_X)\ ,\ A(\mathsf{cons}_X)\ \rangle\ } & A[X] \end{array}$$

Example 3.5. Continuing the example of the representable lax monoidal functor $\mathrm{Hom}(E, -)$, we have that the sequence map is given by evaluating each function in a list:

$$\begin{aligned} \mathsf{seq}_X : [\mathrm{Hom}(E, X)] &\to \mathrm{Hom}(E, [X]) \\ [\phi_1, \ldots, \phi_n] &\mapsto [e \mapsto [\phi_1(e), \ldots, \phi_n(e)]] \end{aligned}$$

In this case, the sequence map is injective, and its image consists of all maps $\Phi : E \to [X]$ such that the composition $\mathsf{len} \circ \Phi : E \to \mathbb{N}$ is constant.

Remark 3.6. In appendix B, we propose a generalization of the sequence map to a natural transformation $F \circ A \to A \circ F$ to a class of functors F beyond the list endofunctor.

3.3. **Alternative formulation.** We give an alternative formulation of the sequence map. For $n \in \mathbb{N}$, define natural transformations

$$\alpha_X^{(n)} : (AX)^n \to A(X^n)$$

(natural in X) recursively as follows. For $n = 0$, we set $\alpha_X^{(0)} = \psi : 1 \to A1$, while for $n \geq 1$, we set $\alpha_X^{(n)}$ to be the following composition:

$$\alpha_X^{(n)} = \left(AX = AX \times (AX)^{n-1} \xrightarrow{1 \times \alpha_X^{(n-1)}} AX \times A(X^{n-1}) \xrightarrow{\alpha_{X,X^{n-1}}} A(X \times X^{n-1}) = A(X^n) \right)$$

In particular, $\alpha_X^{(1)} = \alpha_X$. The sequence map seq_X is then equal to the composition:

$$[AX] = \textstyle\sum_{n=0}^{\infty} (AX)^n \xrightarrow{\sum_n \alpha_X^{(n)}} \sum_{n=0}^{\infty} A(X^n) \xrightarrow{\langle A(\mathrm{inc}_n) | n \in \mathbb{N} \rangle} A[X]$$

where the first map applies $\alpha_X^{(n)}$ to the n-th cofactor, while the second applies $A(\mathrm{inc}_n) : A(X^n) \to A[X]$ to the n-th cofactor, with $\mathrm{inc}_n : X^n \to [X]$ denoting the obvious inclusion. In other words, given a list $\mathbf{p} \in (AX)^n$ of length n with entries in AX, we first compute $\alpha_X^{(n)}(\mathbf{p}) \in A(X^n)$, and then feed that into $A(\mathrm{inc}_n) : A(X^n) \to A[X]$.

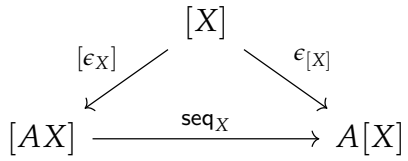
3.4. **Epsilon, Beta, and Gamma.** Let (A, ψ, α) be a lax monoidal functor. Define a natural transformation:

$$\epsilon_X : X \to AX, \qquad x \mapsto A[1 \mapsto x] \circ \psi(1)$$

(natural in X). Alternatively, ϵ_X is defined via the composition:

$$X \simeq \mathrm{Hom}(1, X) \xrightarrow{F} \mathrm{Hom}(F1, FX) \xrightarrow{\psi^*} \mathrm{Hom}(1, FX) \simeq FX$$

In appendix C, we prove that ϵ and the sequence map are related as follows:



It is easy to verify that $A(\mathsf{nil}_X) \circ \psi = \epsilon_{[X]} \circ \mathsf{nil}_X$, so that seq_X takes the empty list in $[AX]$ to the unit $\epsilon_{[X]}$ applied to the empty list in $[X]$, that is, $\mathsf{seq}_X([\,]) = \epsilon_{[X]}([\,])$.

Next, define a natural transformation:

$$\beta_{X,Y,Z} : \mathrm{Hom}(X \times Y, Z) \to \mathrm{Hom}(AX \times AY, AZ)$$
$$f \mapsto Af \circ \alpha_{X,Y}$$

(natural in X, Y, Z). The natural transformation α in the definition of an applicative can be recovered from the natural transformation β by taking $Z = X \times Y$ and $f = \mathrm{id}_{X \times Y}$. Thus, a lax monoidal functor can be defined in terms of the natural transformation β,

with the axioms adapted appropriately. Moreover, there is another natural transformation the can be used to characterize a lax monoidal functor instead of α, namely:

$$\begin{aligned}\gamma_{X,Y} : A\mathrm{Hom}(X,Y) &\to \mathrm{Hom}(AX, AY)\\ \tilde{f} &\mapsto \left[\tilde{x} \mapsto A(\mathrm{ev}) \circ \alpha_{\mathrm{Hom}(X,Y),X}(\tilde{f}, \tilde{x})\right]\end{aligned}$$

In other words, $\gamma_{X,Y}$ can be defined via the hom-product adjunction and the following composition:

$$A\mathrm{Hom}(X,Y) \times AX \xrightarrow{\alpha_{\mathrm{Hom}(X,Y),X}} A(\mathrm{Hom}(X,Y) \times X) \xrightarrow{A(\mathrm{ev})} AY$$

The natural transformation $\alpha_{X,Y}$ is the image of the identity morphism under:

$$\begin{aligned}\mathrm{Hom}(X \times Y, X \times Y) &\xrightarrow{\sim} \mathrm{Hom}(X, \mathrm{Hom}(Y, X \times Y)) \xrightarrow{A} \mathrm{Hom}(AX, A\mathrm{Hom}(Y, X \times Y))\\ &\xrightarrow{\gamma_{Y,X\times Y} \circ -} \mathrm{Hom}(AX, \mathrm{Hom}(AY, A(X \times Y)))\\ &\xrightarrow{\sim} \mathrm{Hom}(AX \times AY, A(X \times Y))\end{aligned}$$

As a formula, we have:

$$\alpha_{X,Y}(\tilde{x}, \tilde{y}) = (\gamma_{Y,X\times Y} \circ A\eta(\tilde{x}))(\tilde{y})$$

where $\eta : X \to \mathrm{Hom}(Y, X \times Y)$ is the map[1] taking x to $y \mapsto (x, y)$.

Example 3.7. For the representable applicative $A = \mathrm{Hom}(E, -)$, the map $\epsilon_X : X \to \mathrm{Hom}(E, X)$ sends $x \in X$ to the constant function at x. Meanwhile, $\beta_{X,Y,Z}$ sends $f : X \times Y \to Z$ to the function $\mathrm{Hom}(E, X) \times \mathrm{Hom}(E, Y) \to \mathrm{Hom}(E, Z)$ given by: $(\phi, \psi) \mapsto [e \mapsto f(\phi(e), \psi(e))]$. For the map γ, we have:

$$\begin{aligned}\gamma_{X,Y} : \mathrm{Hom}(E, \mathrm{Hom}(X,Y)) &\to \mathrm{Hom}(\mathrm{Hom}(E,X), \mathrm{Hom}(E,Y))\\ \Theta &\mapsto [\phi \mapsto [e \mapsto \Theta(e)(\phi(e))]]\end{aligned}$$

As we discuss below, the implementation of an applicative in Haskell uses the analogues of β and γ rather than α.

4. Monads

Once again, unless specified otherwise, all endofunctors are on the category Set of sets.

4.1. **Basics.** Recall that a monad is an endofunctor T, together with unit and multiplication natural transformations, $\epsilon : 1 \to T$ and $\mu : T^2 \to T$, which satisfy the associativity and unit axioms:

$$\mu_X \circ \mu_{TX} = \mu_X \circ T\mu_X \qquad \mu_X \circ \epsilon_{TX} = \mathrm{id}_{TX} = \mu_X \circ T\epsilon_X$$

[1] To verify the formula, one uses the fact that $\alpha_{\mathrm{Hom}(Y,X\times Y),Y} \circ (A\eta \times 1) = A(\eta \times 1) \circ \alpha_{X,Y}$.

Example 4.1. Our running example of the representable endofunctor $\mathrm{Hom}(E, -)$ is a monad, where the unit takes $x \in X$ to the constant map at x in $\mathrm{Hom}(E, X)$, while the multiplication map $\mathrm{Hom}(E, \mathrm{Hom}(E, X)) \to \mathrm{Hom}(E, X)$ can be described as applying the product-Hom adjunction and then pulling back along the diagonal map:

$$\mathrm{Hom}(E, \mathrm{Hom}(E, X)) \xrightarrow{\sim} \mathrm{Hom}(E \times E, X) \xrightarrow{\Delta^*} \mathrm{Hom}(E, X)$$

In other words, $\Phi : E \to \mathrm{Hom}(E, X)$ is sent to the map taking e to $\Phi(e)(e) \in X$.

4.2. **The Kleisli category.** Let T be a monad on the category Set[2]. The Kleisli category Set_T corresponding to T is defined as having the same objects as Set, with morphisms given by:

$$\mathsf{Set}_T(X, Y) = \mathsf{Set}(X, TY)$$

The identity in $\mathsf{Set}_T(X, X)$ is given by the unit $\epsilon_X : X \to TX$, and the composition of $f_1 : X \to TY$ and $f_2 : Y \to TZ$ is given by:

$$f_2 \circ_T f_1 := \mu_Z \circ Tf_2 \circ f_1 = (X \xrightarrow{f_1} TY \xrightarrow{Tf_2} T^2Z \xrightarrow{\mu_Z} TZ)$$

The associativity of composition, as well as the identity properties, follow from (and are in fact equivalent to) the associativity and unit axioms of the monad T. There are functors:

$$\mathsf{Set} \underset{U_T}{\overset{F_T}{\rightleftarrows}} \mathsf{Set}_T$$

defined as follows:

$$F_T X = X \qquad\qquad U_T X = TX$$

$$F_T(X \xrightarrow{f} Y) = X \xrightarrow{f} Y \xrightarrow{\epsilon_Y} TY \qquad\qquad U_T(X \xrightarrow{g} TY) = TX \xrightarrow{Tg} T^2Y \xrightarrow{\mu_Y} TY$$

In other words, F_T is the identity on objects and post-composes with the unit on morphisms: $F_T f = \epsilon_Y \circ f$. Meanwhile, U_T applies T to objects and, on morphisms, applies T and then multiplies: $U_T g = \mu_Y \circ Tg$.

Lemma 4.2. *We have:*

(1) *The functors above form an adjoint pair (F_T, U_T):*

$$\mathsf{Set}_T(F_T X, Y) = \mathsf{Set}_T(X, Y) = \mathsf{Set}(X, TY) = \mathsf{Set}(X, U_T Y).$$

(2) *The monad of the adjunction is $T = U_T \circ F_T$. In particular,*

$$Tf = U_T(\epsilon_Y \circ f), \qquad \mu_X = U_T(\mathrm{id}_{TX}), \qquad \epsilon_X = F_T(\mathrm{id}_X).$$

(3) *The functor U_T applied to morphisms defines a natural transformation*

$$\mathsf{Set}(-, T-) \to \mathsf{Set}(T-, T-)$$

of functors $\mathsf{Set}^{\mathsf{op}} \times \mathsf{Set} \to \mathsf{Set}$.

[2]The discussion of this subsection holds for a monad on any category $\mathcal{C}$

(4) *For any* $g = g_1 : X \to TY$ *and* $g_2 : Y \to TZ$, *we have:*

$$(U_T g) \circ \epsilon_X = g \qquad U_T(\epsilon_X) = \mathrm{id}_{TX} \qquad U_T(U_T(g_2) \circ g_1) = U_T(g_2) \circ U_T(g_1)$$

Sketch of proof. The first claim follows from definitions. The second also follows from definitions, together with the unit axiom to show that $U_T(\epsilon_Y \circ f) = \mu_Y \circ T\epsilon_Y \circ Tf = Tf$. The third claim is a consequence of the facts that (1) U_T is a functor, (2) U_T applies T to objects, and (3) $\mathsf{Set}_T(X, Y) = \mathsf{Set}(X, TY)$. The identities of the final claim are easily verified using (1) the definition that $U_T g = \mu_Y \circ Tg$, (2) the fact that μ and ϵ are natural transformations, and (3) the unit axiom for T. □

Remark 4.3. There is a precise sense in which the Kleisli category, equipped with the adjunction (F_T, U_T) is initial among all adjunctions giving rise to the monad T. The category of T-algebras (also known as the Eilenberg–Moore category) is final.

Definition 4.4. Given a monad T and sets X and Y, the Kleisli composition is the map:

$$TX \times \mathrm{Hom}(X, TY) \to TY$$

taking $(\tilde{x}, g)$ to $\mu_Y \circ Tg(\tilde{x}) = U_T(\tilde{x})$.

Example 4.5. We continue the running example of the representable monad $\mathrm{Hom}(E, -)$. To describe the Kleisli composition, we make use of the product-Hom adjunction:

$$\begin{aligned} TX \times \mathrm{Hom}(X, TY) &= \mathrm{Hom}(E, X) \times \mathrm{Hom}(X, \mathrm{Hom}(E, Y)) \\ &\simeq \mathrm{Hom}(E, X) \times \mathrm{Hom}(E, \mathrm{Hom}(X, Y)) \\ &\simeq \mathrm{Hom}(E, X \times \mathrm{Hom}(X, Y)) \\ &\to \mathrm{Hom}(E, Y) \end{aligned}$$

where the third line uses the fact that a morphism into a product is the same as a pair of morphisms into each factor, and the last map is post-composition with the evaluation map $X \times \mathrm{Hom}(X, Y) \to Y$. More explicitly, given $\phi \in \mathrm{Hom}(E, X)$ and $\Theta \in \mathrm{Hom}(X, \mathrm{Hom}(E, Y))$, we can produce the function $E \to Y$ taking e to $\Theta(\phi(e))(e)$.

4.3. **Monads as lax monoidal functors.** Recall from Section 2.2 that for any endofunctor F on Set we have a natural transformation

$$\mathrm{Hom}(X \times Y, Z) \to \mathrm{Hom}(FX \times FY, F^2 Z)$$

(natural in X, Y, Z). Now suppose $F = T$ is a monad. Post-composing with the multiplication map $\mu : T^2 \to T$, we obtain a natural transformation:

$$T^{(2)} = T^{(2)}_{X,Y,Z} : \mathrm{Hom}(X \times Y, Z) \to \mathrm{Hom}(TX \times TY, TZ) \tag{4.1}$$

We have a number of ways to express this as a formula:

$$\begin{aligned} T^{(2)}(f)(p, q) &= \mu_Z \circ T[x \mapsto Tf(x, -)(q)](p) \\ &= U_T(x \mapsto Tf(x, -)(q))(p) \\ &= U_T(x \mapsto U_T(\epsilon_Z \circ f(x, -))(q))(p) \end{aligned}$$

Set

$$\alpha_{X,Y} := T^{(2)}_{X,Y,X\times Y}(\mathrm{id}_{X\times Y}) : TX \times TY \to T(X \times Y)$$

Then one can show that the triple (T, ϵ_1, α) satisfies to axioms of a lax monoidal functor, and that $\beta = T^{(2)}$. Thus, every monad can be regarded as a lax monoidal functor. As a consequence, we have a sequence natural transformation for monads: $\mathsf{seq} : [T-] \to T[-]$.

However, not all lax monoidal functors are monads. We delay a detailed discussion of examples until Section 6; for now we illustrate a way to check whether a lax monoidal functor is a monad. Given a lax monoidal functor (A, ψ, α), let $\mu : A^2 \to A$ be a natural transformation. Then μ defines a monad structure compatible with the applicative structure if and only if:

(1) the natural transformation $T^{(2)}$ (defined as in Equation 4.1) coincides with the natural transformation β (defined as in Definition 3.1).
(2) μ satisfies associativity and the unit axioms.

5. Haskell

5.1. **Haskell types.** Haskell is a strongly-typed language, which means that every Haskell value and expression has a type. Additionally, to accommodate non-terminating functions, every type is extended by adding a special value called the *bottom*; the resulting category is known as Hask. There are numerous complications when developing mathematical results with the Hask. At the same time:

> From the pragmatic point of view, it's okay to ignore non-terminating functions and bottoms, and treat Hask as bona fide Set. [Mil19]

Hence, in these notes, we model Haskell types with the category Set of sets. Given types `a` and `b` we have the function type `a->b`. The type `a->b->c` is the same as the type `a->(b->c)`. This reflects product-hom adjunction. More generally, the following types are the same:

```
a->b->c->d    a->(b->c->d)    a->(b->(c->d))    a->b->(c->d)
```

5.2. **Type constructors and functors.** Haskell features type constructors that take a type and produce a new type. Examples include the `Maybe` constructor, the `List` constructor[3], and the binary tree constructor `BTree`:

```
data Maybe a = Nothing | Just a
data List a = Nil | Cons a (List a)
data BTree a = Empty | Node a (BTree a) (BTree a)
```

A type constructor may also take in multiple types to produce the new type, as in:

```
data Either a b = Left a | Right b
data ListEither a b = Nil | Cons1 a (ListEither a b) | Cons2 b (
   ListEither a b)
```

[3]The list type with entries of type `a` is usually denoted `[a]`.

In these notes, we focus on single-variable type constructors. A type constructor is a `Functor` in Haskell if it is equipped with a way of producing a new function for every function between the type variables. The minimal complete definition is:

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

Alternatively, we can use infix notation:

```
  <$> :: (a -> b) -> f a -> f b
  p <$> z = fmap p z
```

To connect with category theory, a `Functor` in the sense of Haskell reflects an endofunctor of Set this category if it satisfies the identity and composition axioms:

```
fmap id = id                          -- identity axiom
fmap (f . g)  ==  fmap f . fmap g     -- composition axiom
```

Functors defined in Haskell are expected to respect identities and composition, though there is no way to enforce this in Haskell. Conversely, an endofunctor F on Set defines a natural transformation $\mathrm{Hom}(-,-) \to \mathrm{Hom}(F-,F-)$ of functors $\mathrm{Set}^{\mathrm{op}} \times \mathrm{Set} \to \mathrm{Set}$; this corresponds to `fmap`. The `Maybe`, `List`, and `BTree` constructors are "naturally" functors.

5.3. **Applicatives.** The analogue of lax monoidal functors in Haskell are applicatives. The applicative type constructor class is often defined as:

```
class Functor f => Applicative f where
    pure  :: a -> f a
    (<*>) :: f (a -> b) -> f a -> f b
```

so that `pure` corresponds to the unit ϵ and `<*>` corresponds to the map γ (see Section 3.4). The functor structure map `fmap` can be recovered as:

```
fmap :: (a -> b) -> f a -> f b
fmap phi z = (pure p) <*> z
```

We also have the analogue of β given by:

```
liftA2 :: (a -> b -> c) -> f a -> f b -> f c
liftA2 p z y = p <$> x <*> y
```

More generally, we can lift a map of any number of variables. In pseudo-Haskell:

```
liftAn :: (a1 -> a2 -> ... -> an -> b) -> f a1 -> f a2 -> ... -> f an
   -> f b
liftA2 p z1 z2 ... zn = p <$> z1 <*> z2 <*> ... <*> zn
```

The analogue of the map α does not appear in the official documentation, but can be defined as:

```
(>*<) :: f a -> f b -> f (a, b)
(>*<) z w = (pure p) <*> z <*> w
  where p x y = (x,y)
```

The operator <*> can be recovered from either `liftA2` or >*< via:

```
(<*>) = liftA2 id
(<*>) = fmap ($) . (>*<)
```

For applicatives, the sequence map is denoted `sequenceA`, and can be defined as:

```
sequenceA :: Applicative => [f a] -> f [a]
sequenceA [] = pure []
sequenceA (p:ps) = fmap cons (p >*< (sequenceA ps))
```

5.4. **Monads.** A type constructor `m` is considered a `Monad` in Haskell if it is equipped with (1) a way of extending any function `a -> m b` to a function `m a -> m b`, and (2) a map from any type `a` to the output type `m a`. The minimal complete definition is:

```
class Monad m where
  (>>=)  :: m a -> (  a -> m b) -> m b
  return ::   a                 -> m a
```

A monad in Haskell is expected to satisfy the following identities (which cannot actually be enforced in Haskell):

```
-- x::a     z::m a     f::a -> m b     g::b -> m c
return x >>= f                  =  f x              -- left identity
z        >>= return             =  z                -- right identity
z        >>= (\x -> f x >>= h)  =  (z >>= f) >>= h  -- associativity
```

Moreover, a `Monad` in Haskell is in particular a `Functor`:

```
liftM :: Monad m => (a -> b) -> (m a -> m b)
liftM f z = z >>= (return . f)
```

The identity and composition laws for the `liftM` follow from the identity and associativity laws for the monad `m`. Table 1 gives a dictionary between constructions and identities in Haskell and concepts in category theory in the context of monads.

It is easy to see from Lemma 4.2 that a monad in the sense of category theory defines a `Monad` in the sense of Haskell. In Appendix A, we give a proof of the converse: a `Monad` in Haskell gives rise to a ordinary category theory monad as long as the identity and associativity axioms hold.

We give an implementation of the composition rule mentioned in Section 1:

```
compose :: (a -> m b) -> (b -> m c) -> a -> m c
compose g1 g2 x = (g1 x) >>= g2
```

Haskell	Category Theory
Monad `m`	Monad $T : \mathrm{Set} \to \mathrm{Set}$
`f::a->m b`	$f : X \to TY$
`z >>= f`	$U_T(f)(z)$
`return::a->m a`	$\epsilon : 1 \to T$
`prod::m(ma)->ma, (p -> p >>= id)`	$\mu_X : T^2X \to TX$
`return x >>= f = f x`	$(U_T f) \circ \epsilon_X = f$
`z >>= return = z`	$U_T(\epsilon_X) = \mathrm{id}_{TX}$
`z>>=(\x->f x>>=h) = (z>>=f)>>=h`	$U_T(U_T(h) \circ f) = U_T(h) \circ U_T(f)$
`liftM f z = z >>= (return . f)`	$Tf = U_T(\epsilon_Y \circ f)$

FIGURE 1. Dictionary between Haskell and category theory in the context of monads.

Remark 5.1. Haskell also has an operator `(>>):ma->mb->mb` for a monad `m`. This can be recovered from `(>>=)` by using a constant function in the second factor:

```
z>>w  = z>>= (\x -> w)
```

5.5. **`do` blocks.** Haskell features `do` blocks as a way to abbreviate compositions in the Kleisli category and the functor U_T. To illustrate the motivation, let `z:ma` and `w:mb`. We will often want to perform computations like:

```
z >>= [\x -> w >>= [\y -> u x y]]
```

where `z::ma`, `w::mb`, and `u` is a function of type `a->b->mc`. Walking through this: given `x::a`, we have the function `b->mc` taking `y` to `u x y`. This function feeds in as the second input of the operator `>>=` with `w::mb` being the first input. The operator returns an output of type `mc`. Recalling that `x::a` was fixed, we see that we obtain a function `a->mc`. This function in turn feeds as the second input of the operator `>>=` with `z::ma` being the first input. Hence, we the final output is of type `mc`.

Using `do` blocks, this simplifies to:

```
do
  x <- z
  y <- w
  u x y
```

In general, a code block of the form

```
do
  x <- z
  ...
```

is equivalent to:

```
z >>= [\x -> do ...]
```

Notes:

- The final line of a `do` block must be an expression of type `m a` for some type `a`. One can think of the type of the entire `do` block as having the type of the last line.
- A non-final line may be of the form `v <- expr`, where `expr` has type `m a` for some type `a`. The variable `v` is a dummy variable of type `a` used to define a function with input type `a`.
- Alternatively, a non-final line may be consist of just an `expr` of type `m a` for some type `a`. This line can be replaced by `v <- expr` with no change in the functionality of the code block. (This corresponds to defining a constant function and can be avoided with the operator `>>`, see remark 5.1.)
- The remaining lines are `let` statements, which are less relevant for categorical considerations.

Executing a `do` statement amounts to performing a composition of morphisms in the Kleisli category. Conceptually (though not necessarily practically), one can think of going from the last line to the first.

5.6. **The sequence map for monads.** Recall that the sequence function is defined for any applicative. In Haskell, the sequence function seems to be mostly used for monads (the version for applicatives is called `sequenceA`). The idea of the sequence function is to capture the following function concept in (pseudo-)Haskell:

```
sequence :: Monad m => [m a] -> m [a]
sequence [] = return []
sequence [x1,...,xn] = do   x1 <- p1
                            x2 <- p2
                            ...
                            xn <- pn
                            return [x1,x2,...,xn]
```

Obviously, the above doesn't compile; an actual implementation of the sequence map is:

```
sequence :: Monad m => [m a] -> m [a]
sequence [] = return []
sequence (p:ps) = p >>= \x -> (sequence ps) >>= \ys -> return (x:ys)
```

One can also implement the map $T^{(2)}$, as well as nil and cons:

```
T2 :: (a -> b -> c) -> m a -> m b -> m c
T2 f z u = z >>= (\x -> u >>= return . f x)

nil = [] :: [a]
```

```
cons :: a -> [a] -> [a]
cons x ys = x:ys
```

Taking `b` = `c` = `[a]`, this leads us to implement the sequence function as a fold:

```
sequence :: Monad m => [m a] -> m [a]
sequence = foldr (T2 cons) (return nil)
```

6. Examples

Here are examples of applicatives and monads on the category of sets. For each monad T, we comment on:

- the monad structure maps μ and ϵ,
- the Kleisli composition $TX \times \mathrm{Hom}(X, TY) \to TY$ taking (z, g) to $U_T(g)(z)$,
- the map $T^{(2)} : \mathrm{Hom}(X \times Y, Z) \to \mathrm{Hom}(TX \times TY, TZ)$, and
- the sequence map $\mathsf{seq}_X : [TX] \to T[X]$.

For each applicative A that is not a monad, we comment on:

- the applicative structure maps μ and α (and, if relevant, β and γ),
- the sequence map $\mathsf{seq}_X : [AX] \to A[X]$, and
- the failure of being a monad.

6.1. **The maybe monad.** Consider the functor $TX = 1 + X$, which acts on morphisms by extending a function $f : X \to Y$ to the function $1 + X \to 1 + Y$ sending 1 to 1. This defines a monad with $\epsilon : X \to 1 + X$ including into the second cofactor, and $\mu : T^2X = 1 + 1 + X \to 1 + X$ collapsing both factors of 1 into the single one. The Kleisli composition map:

$$TX \times \mathrm{Hom}(X, TY) = (X + 1) \times \mathrm{Hom}(X, 1 + Y) \to TY = 1 + Y$$

extends a function $X \to 1 + Y$ by sending 1 to 1. The map:

$$T^{(2)} : \mathrm{Hom}(X \times Y, Z) \to \mathrm{Hom}(TX \times TY, TZ) = \mathrm{Hom}(1 + X + Y + X \times Y, 1 + Z)$$

extends a function $X \times Y \to Z$ by sending the cofactors 1, X, and Y to 1. More generally, to describe the sequence map, let $[k_1, \ldots, k_n]$ be a list with entries in $1 + X$, so that each k_i is either equal to 1 or belongs to X. Then:

$$\mathsf{seq}([k_1, \ldots, k_n]) = \begin{cases} [x_1, \ldots, x_n] \in [X] \subseteq 1 + [X] & \text{if } k_i = x_i \in X \text{ for all } i \\ 1 \in 1 + [X] & \text{otherwise} \end{cases}$$

In other words, if any of the k_i's is equal to one, then the output of the sequence map is 1, and otherwise the sequence map is the identity.

6.2. **The overwrite monad.** Consider the endofunctor $TX = X + S$ for a fixed set S. This is a monad with unit being the obvious inclusion $X \to X + S$, while

$$\mu_X : X + S + S \to X + S$$

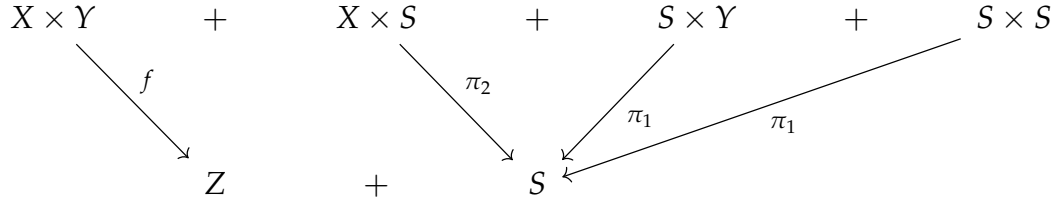
is the identity on X and collapses both S cofactors into the single one using the identity map. The Kleisli composition map

$$TX \times \mathrm{Hom}(X, TY) = (X + S) \times \mathrm{Hom}(X, Y + S) \to TY = Y + S$$

extends a function $X \to Y + S$ by sending S to S identically. The map:

$$T^{(2)} : \mathrm{Hom}(X \times Y, Z) \to \mathrm{Hom}(TX \times TY, TZ) = \mathrm{Hom}((X + S) \times (Y + S), Z + S)$$

takes $f : X \times Y \to Z$ to the map:

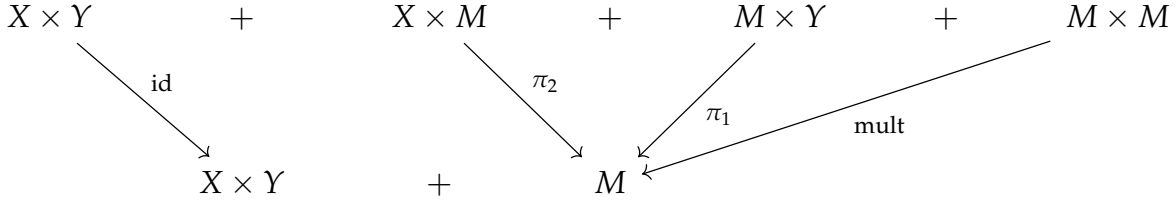


noting that $(X + S) \times (Y + S) = X \times Y + X \times S + S \times Y + S \times S$. We think of the overwrite monad as a generalization of the maybe monad, where failure produces an element of S. If we have two failures, the first overwrites the second, hence the name. (One may altenatively choose the opposite convention, where a later failure overwrites an earlier one.) More generally, to describe the sequence map, let $[k_1, \ldots, k_n]$ be a list with entries in $X + S$, so that each k_i is either belongs to X or to S. Then:

$$\mathsf{seq}_X([k_1, \ldots, k_n]) = \begin{cases} [x_1, \ldots, x_n] \in [X] \subseteq S + [X] & \text{if } k_i = x_i \in X \text{ for all } i \\ s = k_i \in S \subseteq S + [X] & \text{where } i \text{ is minimal with } k_i \in S \end{cases}$$

In other words, if any of the k_i's is in S, then the output of the sequence map is the first of these, and otherwise the sequence map is the identity.

6.3. **The validation applicative.** Now consider the endofunctor $AX = X + M$ where M is a monoid. This is an applicative with unit being the obvious inclusion and the map $\alpha : (X + M) \times (Y + M) \to X \times Y + M$ is indicated in the following diagram:



where mult : $M \times M \to M$ is the monadic multiplication. To describe the sequence map, let $[k_1, \ldots, k_n]$ be a list with entries in $X + M$, so that each k_i is either belongs to X or to M. Then, if $k_i = x_i \in X$ for all i, we have:

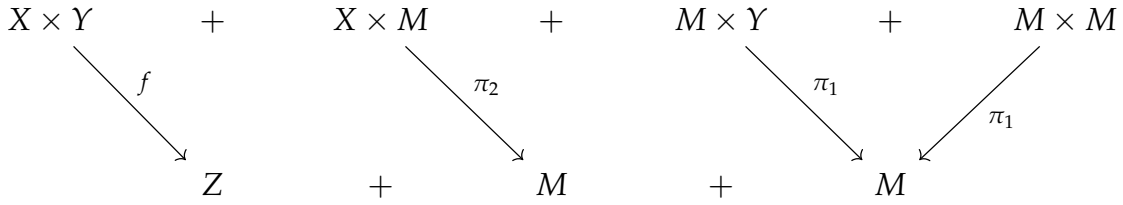
$$\mathsf{seq}_X([k_1, \ldots, k_n]) = [x_1, \ldots, x_n] \in [X] \subseteq M + [X]$$

Otherwise, let $(i_1, \ldots, i_s)$ be the indices, in order, with $k_{i_j} \in M$, and set $m_j = k_{i_j}$. With this notation:

$$\mathsf{seq}_X([k_1, \ldots, k_n]) = m_1 \cdot m_2 \cdot \cdots \cdot m_s \in M \subseteq M + [X]$$

In other words, if any of the k_i's is in M, then the output of the sequence map is the product of these, and otherwise the sequence map is the identity.

We claim that, as long as M is non-trivial, the validation applicative is not a monad. First we compute the map from Equation 2.1 in this case; it takes $f : X \times Y \to Z$ to the map from $(X + M) \times (Y + M)$ to $(Z + M) + M$ indicated by the following diagram:



Now, take a potential monad structure $\mu_Z : Z + M + M \to Z + M$. Compatibility with the unit of the applicative implies that $\mu_Z = \mathrm{id}_Z + \langle \mathrm{id}_M, \mathrm{id}_M \rangle$. On the other hand, compatibility with the diagrams above implies that that the multiplication $M \times M \to M$ coincides with the projection map π_1 onto the first factor. In other words, $1 = \pi_1(1, m) = \mathrm{mult}(1, m) = m$ for all $m \in M$. This is only possible if M is the trivial monoid.

6.4. **The list monad.** The list functor $TX = [X] = \mathsf{List}(X) = \sum_{n=0}^{\infty} X^n$ defines a monad with $\epsilon : X \to [X]$ including X as singleton lists, and $\mu : T^2X \to TX$ concatenating a list of lists into a single list. The Kleisli composition:

$$[X] \times \mathrm{Hom}(X, [Y]) \to [Y]$$

takes a list $[x_1, \ldots, x_n]$ and a function $f : X \to [Y]$ to the concatenation of the lists $f(x_1), ..., f(x_n)$. The map

$$T^{(2)} : \mathrm{Hom}(X \times Y, Z) \to \mathrm{Hom}([X] \times [Y], [Z])$$

takes $f : X \times Y \to Z$ to the function taking the pair of lists $(\mathbf{x}, \mathbf{y})$ to the list consisting of all f evaluated on all combinations of pairs of elements formed by taking one element of $\mathbf{x} = [x_1, \ldots, x_n]$ and one element of $\mathbf{y} = [y_1, \ldots, y_m]$:

$$[f(x_i, y_j) \mid i = 1, \ldots, n; j = 1, \ldots, m]$$

In particular, the length of the output list is the product of the lengths of the input lists. More generally, the sequence map takes a list of lists $[\mathbf{x}_1, \ldots, \mathbf{x}_n]$ to the list of all lists of length n whose i-th entry is an entry of $\mathbf{x}_i$.

$$\mathsf{seq}([\mathbf{x}_1, \ldots, \mathbf{x}_n]) = [[\mathbf{x}_{1,i_1}, \mathbf{x}_{2,i_2}, \ldots, \mathbf{x}_{n,i_n}] \mid i_1 = 1, \ldots, \mathsf{len}(\mathbf{x}_i), \ldots, i_n = 1, \ldots, \mathsf{len}(\mathbf{x}_n)]$$

The number of such lists is the product of the lengths of the $\mathbf{x}_i$. If any of the input lists is empty, then the output list is empty.

Remark 6.1. The maybe monad includes into the list monad via the inclusion $1 + X \hookrightarrow [X]$ as the lists of length at most one. This inclusion commutes with the unit maps.

6.5. **The state monad.** Fix a set S, the "state". The functor

$$TX = \mathrm{Hom}(S, X \times S) \simeq \mathrm{Hom}(S, X) \times \mathrm{End}(S)$$

can be thought of all ways to update the state (the $\mathrm{End}(S)$ factor) and producing a "value" in the set X. It takes $f : X \to Y$ to the map that post-composes with $f \times 1$. This functor defines a monad with $\epsilon : X \to \mathrm{Hom}(S, X \times S)$ taking x to the function $s \mapsto (x, s)$ for all s. The product

$$\mu : X \to \mathrm{Hom}(S, \mathrm{Hom}(S, X \times S) \times S) \to \mathrm{Hom}(S, X \times S)$$

is post-composition with the evaluation map $\mathrm{Hom}(S, X \times S) \times S \to X \times S$. The Kleisli composition:

$$\mathrm{Hom}(S, X \times S) \times \mathrm{Hom}(X, \mathrm{Hom}(S, Y \times S)) \to \mathrm{Hom}(S, Y \times S)$$

is equivalent, via the $(\times, \mathrm{Hom})$ adjunction, to a map

$$\mathrm{Hom}(S, X \times S) \times \mathrm{Hom}(X \times S, Y \times S) \to \mathrm{Hom}(S, Y \times S)$$

and is given by composition. The map

$$T^{(2)} : \mathrm{Hom}(X \times Y, Z) \to \mathrm{Hom}(\mathrm{Hom}(S, X \times S) \times \mathrm{Hom}(S, Y \times S), \mathrm{Hom}(S, Z \times S))$$

takes $f : X \times Y \to Z$ to the function mapping the pair $(\phi : S \to X \times S\ ,\ \psi : S \to Y \times S)$ to the composition:

$$S \xrightarrow{\phi} X \times S \xrightarrow{1 \times \psi} X \times Y \times S \xrightarrow{f \times 1} Z \times S$$

The sequence map $\mathsf{seq}_X : [\mathrm{Hom}(S, X \times S)] \to \mathrm{Hom}(S, [X] \times S)$ takes a list of functions $[\phi_1, \ldots, \phi_n]$ to the function

$$s \mapsto ([x_1, \ldots, x_n], s_n)$$

where the $x_i \in X$ and $s_i \in S$ are defined recursively via:

$$s_0 = s, \qquad \text{and} \qquad (x_i, s_i) = \phi_i(s_{i-1}) \quad \text{for } i = 1, \ldots, n.$$

In other words, the x_i are the values produced while successively updating the state, and s_n is the final state. Equivalently, $\mathsf{seq}_X([]) = [s \mapsto ([], s)]$ and $\mathsf{seq}_X(\phi : \overline{\phi}) = [s \mapsto (x : \mathbf{x}, r)]$ where $\mathsf{seq}_X(\overline{\phi})(s) = (\mathbf{x}, t)$ and $\phi(t) = (x, r)$.

Remark 6.2. The input/output monad `IO` can be regarded as a state monad where the state represents "the real world":

```
IO a  ⇔  theRealWorld -> (a, theRealWorld)
```

This is not how the `IO` monad is actually implemented, but this perspective illustrates how Haskell handles side effects in a purely functional way by modeling them as a change in state. An expression of type `IO a` represents an action that produces a value of type `a` and changes the state of the real world (i.e. has side-effects) without actually implementing this change (hence maintinaing purity).

6.6. **The free semigroup module monad.** Recall that a semiring is defined in the same way as a ring, except without the assumption of additive inverses. Let R be a semiring and consider the functor

$$TX = \mathrm{Hom}_{\mathrm{f}}(X, R),$$

where $\mathrm{Hom}_{\mathrm{f}}$ denotes the set of functions with finite support. This is a covariant functor via pushforward of functions:

$$T(X \xrightarrow{f} Y) : \mathrm{Hom}_{\mathrm{f}}(X, R) \to \mathrm{Hom}_{\mathrm{f}}(Y, R), \qquad \phi \mapsto f_*\phi = \left[y \mapsto \sum_{x \in f^{-1}(y)} \phi(x) \right]$$

Relevant instances of this functor are:

- The natural numbers $\mathbb{N}$. In this case, $TX = \mathsf{Bag}(X)$ is the Bag endofunctor. A bag of elements of X can also be thought of a multi-set; every element has some multiplicity. There is a forgetful natural transformation of monads $\mathsf{List} \to \mathsf{Bag}$.
- The real numbers $\mathbb{R}$ (or any field). In this case, TX is the vector space with basis given by the elements of X.
- The Boolean set $\{0, 1\}$ with OR as addition and AND as multiplication. In this case, $TX = \mathcal{P}_{\mathrm{f}}(X)$ is the set of finite subsets of X.

Observe that TX is the free R-module on the set X, and Tf is a linear map for any $f : X \to Y$. If X is finite, then $TX \simeq R^{|X|}$ is itself a semiring with coordinate-wise operations. If X is not finite, then TX is a semiring without multiplicative unit.

We now discuss the monad structure. The unit $\epsilon_X : X \to \mathrm{Hom}_f(X, R)$ picks out the delta functions, so that $\epsilon_X(x)(x') = 1$ if $x = x'$ and zero otherwise. In other symbols, $\epsilon_X(x) = \delta_x$. The monad product is given by:

$$\mu_X : \mathrm{Hom}_{\mathrm{f}}(\mathrm{Hom}_{\mathrm{f}}(X, R), R) \to \mathrm{Hom}_{\mathrm{f}}(X, R)$$
$$\Phi \mapsto \left[x \mapsto \sum_{\phi \in \mathrm{Hom}_{\mathrm{f}}(X,R)} \Phi(\phi)\phi(x) \right]$$

The sum is well-defined due to the finite support condition. The Kleisli composition is given by:

$$\mathrm{Hom}_{\mathrm{f}}(X, R) \times \mathrm{Hom}(X, \mathrm{Hom}_{\mathrm{f}}(Y, R)) \to \mathrm{Hom}_{\mathrm{f}}(Y, R)$$
$$(\phi, f) \mapsto \left[y \mapsto \sum_{x \in X} \phi(x) f(x)(y) \right]$$

Note that (δ_x, f) is sent to $f(x)$. The Kleisli composition can be thought of as a convolution; let $\pi_1 : X \times Y \to X$ and $\pi_2 : X \times Y \to Y$ be the projection maps. Then (ϕ, f) goes to $(\pi_2)_*(\pi_1^* \cdot f)$, where π_1^* is the pullback, $(\pi_2)_*$ is the pushforward, we regard f as a function $X \times Y \to R$, and $\cdot$ denotes point-wise multiplication.

Moving on, we have:

$$T^{(2)} : \mathrm{Hom}(X \times Y, Z) \to \mathrm{Hom}(\mathrm{Hom}_{\mathrm{f}}(X, R) \times \mathrm{Hom}_{\mathrm{f}}(Y, R), \mathrm{Hom}_{\mathrm{f}}(Z, R))$$
$$f \mapsto \left[(\phi, \psi) \mapsto \left[z \mapsto \sum_{(x,y) \in f^{-1}(z)} \phi(x)\psi(y) \right] \right]$$

Another interpretation: (ϕ, ψ) defines a function $\phi \times \psi : X \times Y \to R \times R$, which we can post-compose with the multiplication map $\mathrm{mult}_R : R \times R$. Then $T^{(2)}f$ sends (ϕ, ψ) to the pushforward $f_*(\mathrm{mult}_R \circ (\phi \times \psi))$. We make two final remarks about $T^{(2)}f$: (1) it is not linear, but bilinear, (2) it sends (δ_x, δ_y) to $\delta_{f(x,y)}$.

Finally, we the sequence map is given by:

$$\mathsf{seq}_X : [\mathrm{Hom}(X, R)] \to \mathrm{Hom}([X], R)$$
$$[\phi_1, \ldots, \phi_n] \mapsto \left[\mathbf{x} \mapsto \begin{cases} \prod_{i=1}^{n} \phi_i(x_i) & \text{if } \mathrm{len}(\mathbf{x}) = n \\ 0 & \text{otherwise} \end{cases} \right]$$

In particular, the sequence map factors through a map $\mathrm{Hom}(X, R)^n \to \mathrm{Hom}(X^n, R)$, and sends a list of delta functions to the delta function on the corresponding list: $[\delta_{x_1}, \ldots, \delta_{x_n}] \mapsto \delta_{[x_1,\ldots,x_n]}$. For $n = 0$, we have that $\mathsf{seq}_X([]) = \delta_{[]}$ is the delta function at the empty set. We revisit the relevant instances of this functor:

- For $R = \mathbb{N}$, the sequence map corresponds to the well-known natural transformation $\mathsf{List} \circ \mathsf{Bag} \to \mathsf{Bag} \circ \mathsf{List}$. Given a list of n bags, the multiplicity of a list in the output bag of lists is the number of ways the list can be realized via taking the first element from the first bag, the second from the second bag, etc. So only lists of length n can have non-zero multiplicity.

- If R is a field, then the sequence map restricts to the natural map from the direct sum of n copies of a vector space to the tensor product of n copies of the same vector space: $V^{\oplus n} \to V^{\otimes n}$ taking $(v_1, \ldots, v_n)$ to $v_1 \otimes \cdots \otimes v_n$.

- Finally, for the finite power set monad, the sequence map takes a list of subsets $[S_1, \ldots, S_n]$ of X to the subset of $[X]$ consisting of all lists $[x_1, \ldots, x_n]$ where x_i belongs to S_i.

Remark 6.3. The power set monad $TX = \mathcal{P}(X)$ under direct images does not exactly fit into the set of examples discussed in this section since X may be infinite. However, the same formulas still hold. In particular, the sequence map is defined in exactly the same way as the sequence map for the finite power set monad.

Remark 6.4. The Giry monad is a monad on the category of measurable spaces that sends a measurable space to the set of probability measures on that space, which is itself a measurable space in a natural way. While it cannot be strictly regarded as "free semigroup module monad", there are similarities: pushforward of measures is used to define the functor structur, the unit is given by delta functions, and the multiplication is given by taking expected value using a formula similar to that of μ_X above.

6.7. **The representable monad.** Let E be a set. We have already discussed the representable enofunctor $TX = \mathrm{Hom}(E, -)$ in detail. Here we note a few additional observations. The map $T^{(2)}$ can be described as:

$$T^{(2)} : \mathrm{Hom}(X \times Y, Z) \to \mathrm{Hom}(\mathrm{Hom}(E, X) \times \mathrm{Hom}(E, Y), \mathrm{Hom}(E, Z))$$
$$f \mapsto [(\phi, \psi) \mapsto f \circ \Delta^*(\phi \times \psi)]$$

So that $T^{(2)} f(\phi, \psi)$ sends e to $f(\phi(e), \psi(e))$. We have shown that the sequence map sends $[\phi_1, \ldots, \phi_n] \in [\mathrm{Hom}(E, X)]$ to the map sending e to $[\phi_1(e), \ldots, \phi_n(e)] \in [X]$. One can describe this map as falling out of properties of how product and coproducts interact with Hom sets:

$$\mathsf{seq}_X : [\mathrm{Hom}(E, X)] = \sum_{n=0}^{\infty} \mathrm{Hom}(E, X)^n = \sum_{n=0}^{\infty} \mathrm{Hom}(E, X^n)$$
$$\to \mathrm{Hom}(E, \sum_{n=0}^{\infty} X^n) = \mathrm{Hom}(E, [X])$$

6.8. **ZipList.** The Haskell type constructor `ZipList` corresponds to the endofunctor sending a set X to the set of all lists with entries in X, where each list can be finite or infinite. Given a map $f : X \to Y$, the corresponding map on ziplists applies f entry-wise:

$$\mathtt{ZipList}(f)([x_1, x_2, \ldots]) = [f(x_1), f(x_2), \ldots]$$

In particular, $\mathtt{ZipList}(f)$ preserves lengths. The endofunctor `ZipList` is an applicative with $\psi : 1 \to \mathtt{ZipList}(1)$ selecting the unique infinite list. Consequently, we have:

$$\epsilon_X : X \to \mathtt{ZipList}(X); \qquad x \mapsto [x, x, x, ...]$$

The map α is given by taking the diagonal:

$$\mathtt{ZipList}(X) \times \mathtt{ZipList}(Y) \to \mathtt{ZipList}(X \times Y)$$
$$\mathbf{x}, \mathbf{y} \mapsto [(x_1, y_1), (x_2, y_2), (x_3, y_3) \ldots]$$

The sequence map $\mathsf{seq}_X : [\mathtt{ZipList}(X)] \to \mathtt{ZipList}([X])$ is given by:

$$[\mathbf{x}_1, \mathbf{x}_2, \ldots] \mapsto [[\mathbf{x}_1[1], \mathbf{x}_2[1], \mathbf{x}_3[1], \ldots]$$
$$[\mathbf{x}_1[2], \mathbf{x}_2[2], \mathbf{x}_3[2], \ldots]$$
$$[\mathbf{x}_1[3], \mathbf{x}_2[3], \mathbf{x}_3[3], \ldots]$$
$$\ldots]$$

so that the first entry of the output list is the list of first elements, the second is the list of second elements, and so on. The length of the output list is the minimum length of the lists in the input list.

We henceforth abbreviate `ZipList` by just A, and argue that there is no monad structure on A compatible with the applicative structure. To this end, we first regard A as an

endofunctor and compute the map of Equation 2.1 as taking $f : X \times Y \to Z$ to the map $AX \times AY \to A^2Z$ defined as:

$$\begin{aligned}(\mathbf{x}, \mathbf{y}) \mapsto [&[f(x_1, y_1), f(x_1, y_2), f(x_1, y_3) \dots] \\ &[f(x_2, y_1), f(x_2, y_2), f(x_2, y_3) \dots] \\ &[f(x_3, y_1), f(x_3, y_2), f(x_3, y_3) \dots] \\ &\dots]\end{aligned}$$

where $\mathbf{x} = [x_1, x_2, x_3, \dots]$ and $\mathbf{y} = [y_1, y_2, y_3, \dots]$. Now suppose we have a natural transformation $\mu : A^2 \to A$, such that the map $AX \times AY \to A^2Z$ post-composed with μ_Z recovers the applicative structure $AX \times AY \to AZ$. By inspection of the above formulas, the only candidate map $\mu_Z : A^2Z \to AZ$ is the "diagonal" map that takes the first element of the first list, the second element of the second list, etc., stops at the k-th list if the length of the $(k+1)$-st list is less than $k+1$.

However, this operation is not associative. For example, take $Z = \{1\}$ and consider the element

$$[[[1]], [[], [1, 1]]] \in A^3\{1\}.$$

We have:

$$\mu_Z \circ \mu_{AZ}([[[1]], [[], [1, 1]]]) = \mu_Z([[1], [1, 1]]) = [1, 1]$$

but:

$$\mu_Z \circ A\mu_Z([[[1]], [[], [1, 1]]]) = \mu_Z([[1], []]) = [1]$$

Remark 6.5. We note that there are natural transformations:

$$\mathrm{Hom}(\mathbb{N}, -) \to \texttt{ZipList} \to \mathrm{Hom}(\mathbb{N}, \mathsf{Maybe}(-))$$

which (1) are injective on any set X, (2) commute with the unit maps, and (3) are compatible with the applicative structures. The first picks out the infinite lists of `ZipList`, while the second can be described as on a set X as:

$$\begin{aligned}\texttt{ZipList}(X) &\to \mathrm{Hom}(\mathbb{N}, 1 + X) \\ \mathbf{x} &\mapsto \left[n \mapsto \begin{cases} x_n & \text{if } n \leq \mathrm{len}(\mathbf{x}) \\ 1 & \text{otherwise} \end{cases} \right]\end{aligned}$$

One can verify that $\mathrm{Hom}(\mathbb{N}, \mathsf{Maybe}(X))$ is indeed a monad, and that its multiplication does not preserve `ZipList`.

6.9. **Summary.** The following two diagrams illustrate the interrelations between the examples we have considered.

$$\begin{array}{ccccccccccc} X & \hookrightarrow & \mathsf{Maybe}(X) & \hookrightarrow & \mathsf{List}(X) & \twoheadrightarrow & \mathsf{Bag}(X) & \twoheadrightarrow & \mathcal{P}_f(X) & \hookrightarrow & \mathcal{P}(X) \\ & & \downarrow & & & & & & & & \\ & & X + M & & & & & & & & \end{array}$$

(where M is a monoid in Set)

$$\begin{array}{ccccccc} X & \xrightarrow{\sim} & \mathrm{Hom}(1,X) & \hookrightarrow & \mathrm{Hom}(\mathbb{N},X) & \hookrightarrow & \mathrm{Hom}(\mathbb{N},\mathsf{Maybe}(X)) \\ & & & & \searrow \quad \nearrow & & \\ & & & & \mathtt{ZipList}(X) & & \end{array}$$

In both diagrams, the composition from X to any of the items is the unit map, and all items in the top row are monads, while $X + M$ and $\mathtt{ZipList}(X)$ are only applicatives. Without being precise, we also note that there are "maps of monads"

$$\mathrm{Hom}_f(X, R_1) \to \mathrm{Hom}_f(X, R_2), \qquad \mathrm{Hom}(E_2, X) \to \mathrm{Hom}(E_1, X)$$

for any map of semirings $R_1 \to R_2$ and any map of sets $E_1 \to E_2$. In particular, there is a canonical map $\mathsf{Bag}(X) \to \mathrm{Hom}_f(X, R)$ for any semiring R and, for any set E, the unit $X \to \mathrm{Hom}(E, X)$ is pullback along the unique map $E \to 1$. Finally, we note that there is a natural transformation from the representable monad $\mathrm{Hom}(S, -)$ to the state monad $\mathrm{Hom}(S, - \times S)$ sending $\phi : S \to X$ to the map $s \mapsto (\phi(s), s)$.

7. Exercises

(1) Let $F : \mathcal{C} \to \mathcal{D}$ be a functor. Show that the application of F to morphisms defines a natural transformation $\mathrm{Hom}(-,-) \to \mathrm{Hom}(F-, F-)$ of functors $\mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathsf{Set}$.

(2) Let (F, ϵ, α) be a lax monoidal functor on Set. Prove that the following diagram commutes:

$$\begin{array}{ccccc} \mathrm{Hom}(X,Y) \times FX & \xrightarrow{\epsilon_{\mathrm{Hom}(X,Y)} \times \mathrm{id}} & F\mathrm{Hom}(X,Y) \times FX & \xrightarrow{\alpha_{\mathrm{Hom}(X,Y),X}} & F(\mathrm{Hom}(X,Y) \times X) \\ \downarrow{\scriptstyle F \times \mathrm{id}} & & & & \downarrow{\scriptstyle F(\mathrm{ev})} \\ \mathrm{Hom}(FX, FY) \times FX & & \xrightarrow{\qquad\qquad \mathrm{ev} \qquad\qquad} & & FY \end{array}$$

where map from the top left to the bottom left is F applied to morphisms in the first factor. (Suggestion: given $f \in \mathrm{Hom}(X, Y)$, consider the map $1 \to \mathrm{Hom}(X, Y)$ that picks out f. Also use the naturality of ϵ)

(3) Let $F : \mathcal{C} \to \mathcal{D}$ be a functor, and suppose both $\mathcal{C}$ and $\mathcal{D}$ have products and a terminal object.

(a) Define a natural transformation:

$$\mathsf{cart} : F \times (- \times -) \to (- \times_{F(1)} -) \circ (F \times F)$$

of functors $\mathcal{C} \times \mathcal{C} \to \mathcal{D}$. (Hint: use the projection maps $\pi_1 : X \times Y \to X$ and $\pi_2 : X \times Y \to Y$.)

(b) For a monad T, argue that $\mathsf{cart}_{X,Y} \circ \epsilon_{X \times Y} = \epsilon_X \times \epsilon_Y$ as morphisms from $X \times Y$ to $TX \times TY$.

(c) The functor F is said to be *Cartesian* if cart is a natural isomorphism. Argue that the list endofunctor is Cartesian, but the bag endofunctor is not.

(4) Show that requiring the associativity axiom for a monad T is equivalent to requiring the multiplication in the Kleisli category to be associative. Similarly, show that the unit axiom for a monad T is equivalent to the identity properties of $\epsilon_X \in \mathsf{Set}_T(X, X)$ in the Kleisli category.

(5) Completing the discussion of Section 4.3, verify that the associativity and unit axioms of a monad (T, ϵ, μ) translate to the associativity and unit axioms of the proposed lax monoidal functor (T, ϵ, α).

(6) Given a monad T, recursively define $\mu^{(n)} : T^{n+1}X \to TX$ as:

$$\begin{aligned} \mu^{(0)} &= \mathrm{id}_{TX} \\ \mu^{(n)} &= \mu^{(n-1)} \circ \mu_{T^{n-1}X} \qquad \text{for } n \geq 1 \end{aligned}$$

Show that $\mu^{(n)} = \mu^{(n-1)} \circ T\mu_{T^{n-2}X}$ for $n \geq 2$.

(7) Let R be a semiring and let $T = \mathrm{Hom}_{\mathrm{fin}}(-, R)$. Define a map:

$$\begin{aligned} m^{(n)} : X \times TX \times T^2X \times \cdots \times T^nX &\to R \\ (\phi_0, \phi_1, \ldots, \phi_n) &\mapsto \prod_{i=1}^{n} \phi_i(\phi_{i-1}) \end{aligned}$$

Recall the map $\mu^{(n)} : T^{n+1}X \to TX$. Show that:

$$\mu^{(n)}(\Theta)(x) = \sum_{\vec{\phi}} m^{(n+1)}(\Theta, \vec{\phi}, x)$$

where the sum is over all $\vec{\phi} = (\phi_1, \ldots, \phi_n) \in TX \times \cdots \times T^nX$ and $(\Theta, \vec{\phi}, x) = (\Theta, \phi_1, \ldots, \phi_n, x)$.

(8) Let T be a monad. Recall the Cartesian map from Exercise 3; we compose it with the obvious inclusion:

$$T(X \times Y) \xrightarrow{\mathsf{cart}_{X,Y}} TX \times_{T1} TY \xhookrightarrow{\mathrm{inc}} TX \times TY$$

Applying $T^{(2)}_{X,Y}$ and pulling back along this composition, we obtain:

$$\mathrm{Hom}(X \times Y, Z) \xrightarrow{T^{(2)}_{X,Y}} \mathrm{Hom}(TX \times TY, TZ) \xrightarrow{(\mathrm{inc} \circ \mathsf{cart}_{X,Y})^*} \mathrm{Hom}(T(X \times Y), TZ)$$

Prove that, in general, this composition is NOT just the functor T applied to morphisms. For example, take T to be list monad.

(9) Let T be a monad, and let $p \in TX$ for some set X. Show that $\mathsf{seq}([p]) = T[x \to [x]](p)$. In other words, we apply T to the morphism $X \to [X]$ that takes each

element to the singleton list, and evaluate at $p \in TX$. Furthermore, show that $\mathsf{seq}([\epsilon_X x]) = \epsilon_{[X]}[x]$.

(10) Let M be a monoid. Argue that $X \mapsto X \times M$ is a monad. Compute the Kleisli composition and sequence maps.

(11) Let F be an endofunctor on the category of sets.

(a) Generalize the map of Equation 2.1 by induction to a natural transformation

$$\mathrm{Hom}\,(X_1 \times \cdots \times X_n, Z) \to \mathrm{Hom}\,(FX_1 \times \cdots \times FX_n, F^n Z)$$

of functors $(\mathsf{Set}^{\mathrm{op}})^n \times \mathsf{Set} \to \mathsf{Set}$. (There are choices, but proceed left to right.)

(b) In particular, considering the diagonal in $(\mathsf{Set}^{\mathrm{op}})^n$, we have a natural transformation

$$\mathrm{Hom}\,(X^n, Z) \to \mathrm{Hom}\,((FX)^n, F^n Z)$$

of functors $\mathsf{Set}^{\mathrm{op}} \times \mathsf{Set} \to \mathsf{Set}$.

(c) Now suppose $F = T$ is a monad. Using part (c) and $\mu : T^2 \to T$ define a natural transformation:

$$T^{(n)}_{X,Z} : \mathrm{Hom}(X^n, Z) \to \mathrm{Hom}((TX)^n, TZ)$$

with $T^{(0)}_Z : Z \to TZ$ being the unit, $T^{(1)}_{X,Z} : \mathrm{Hom}(X, Z) \to \mathrm{Hom}(TX, TZ)$ being the functor T applied to morphisms, and $T^{(2)}$ coinciding with the definition given above.

(d) Use the $T^{(n)}$ to define a natural transformation:

$$T^{(*)} : \mathrm{Hom}([X], Z) \to \mathrm{Hom}([TX], TZ)$$

Taking $Z = [X]$, argue that $\mathrm{id}_{[X]}$ is mapped to the sequence map seq_X.

(e) Implement $T^{(*)}$ as a function in Haskell via:

```
star :: Monad m => ([a]-> b) -> [m a] -> m b
star f [] = return (f [])
star f (p:ps) = p >>= \x -> star (\ys -> f (x:ys)) ps
```

Argue that the function `star f` is equal to `liftM f . sequence`. In particular, `star id` is equal to `sequence`.

References

[Mil17] Bartosz Milewski, *Applicative Functors*, 2017.

[Mil19] ———, *Category Theory for Programmers by Bartosz Milewski | Blurb Books*, 2019.

[oH] University of Helsinki, *Haskell MOOC*. Accessed: 2025-08-13.

[Rie17] Emily Riehl, *Category Theory in Context*, Courier Dover Publications, 2017.

Appendix A. Characterization of monads

Consider the following data:

- An endofunctor $T : \mathsf{Set} \to \mathsf{Set}$.
- A natural transformation $\epsilon : 1 \to T$ of endofunctors of Set.
- A natural transformation $p : \mathrm{Hom}(-, T-) \to \mathrm{Hom}(T-, T-)$ of functors $\mathsf{Set}^{\mathrm{op}} \times \mathsf{Set} \to \mathsf{Set}$.

For any X, set $\mu_X = p_{TX,X}(\mathrm{id}_{TX}) : T^2X \to TX$. It is easy to see that $\mu : T^2 \to T$ is a natural transformation.

Proposition A.1. *The data (T, ϵ, μ) is a monad if and only if the following identities hold:*

$$p_{XY}(g) \circ \epsilon_X = g \qquad p_{XX}(\epsilon_X) = \mathrm{id}_{TX} \qquad p_{YZ}(g_2) \circ p_{XY}(g_1) = p_{XZ}(p_{YZ}(g_2) \circ g_1)$$

where $g = g_1 : X \to TY$ and $g_2 : Y \to TZ$.

Sketch of proof. The forward implication follows from Lemma 4.2 above, setting p to be the natural transformation on morphisms induced by U_T. For the opposite implication, one must show that the associativity and unit axioms hold. The easiest is the left unit axiom, which follows immediately from definitions and hypotheses:

$$\mu_X \circ \epsilon_{TX} = p_{TX,X}(\mathrm{id}_{TX}) \circ \epsilon_{TX} = \mathrm{id}_{TX}$$

The verification of the right unit axiom is:

$$\begin{aligned} \mu_X \circ T\epsilon_X &= p_{TX,X}(\mathrm{id}_{TX}) \circ T\epsilon_X = p_{X,X}(\mathrm{id}_{TX} \circ \epsilon_X) \\ &= p_{X,X}(\epsilon_X) = \mathrm{id}_{TX} \end{aligned}$$

where the first equality follows from definitions, the second from the fact that p is a natural transformation, and the fourth from the hypotheses. Finally, for associativity:

$$\begin{aligned} \mu_X \circ T\mu_X &= p_{TX,X}(\mathrm{id}_{TX}) \circ T\mu_X = p_{T^2X,X}(\mathrm{id}_{TX} \circ \mu_X) = p_{T^2X,X}(\mu_X) \\ &= p_{T^2X,X}(\mu_X \circ \mathrm{id}_{T^2X}) = \mu_X \circ p_{T^2X,TX}(\mathrm{id}_{T^2X}) = \mu_X \circ \mu_{TX} \end{aligned}$$

where we have twice used the fact that p is a natural transformation. □

Appendix B. Generalized sequence map

B.1. **Introduction.** In this appendix, we give a sketch of a generalization of the sequence map beyond the List inductive type. We assume familiarity with F-algebras and initial algebras.

To warm-up, recall that $\mathsf{List}(X)$ is the initial algebra of the endofunctor $F_X : S \mapsto 1 + X \times S$. Another example to consider is that of binary trees. The set $\mathsf{BTree}(X)$ of binary trees with nodes labeled by X is the initial algebra of the endofunctor $F_X : S \mapsto 1 + X \times S^2$. In both cases, we have the assignment $X \to F_X$ constittutes a functor $\mathsf{Set} \to \mathrm{End}(\mathsf{Set})$.

B.2. **Statement.** Now fix a non-negative integer $d \geq 1$ and consider a functor:

$$\mathsf{Set}^d \to \mathsf{End}(\mathsf{Set}), \qquad \mathbf{X} = (X_1, \ldots, X_d) \mapsto F_{\mathbf{X}}$$

that takes a tuple of sets to an endofunctor of Set. We make the following assumptions:

> *Assumption 1*: The endofunctor $F_{\mathbf{X}}$ has an initial algebra for every $\mathbf{X}$; denote it as $\eta(F; X_1, \ldots, X_d)$. This gives a functor $\eta(F; -) : \mathsf{Set}^d \to \mathsf{Set}$.
>
> *Assumption 2*: For each $\mathbf{X}$ and S, the set $F_{\mathbf{X}}(S)$ is a coproduct of terms of the form $S^{m_0} \times X_1^{m_1} \times \cdots \times X_d^{m_d}$ for some $m_i \in \mathbb{N}$.

Let (A, ϵ, α) be a lax monoidal endofunctor of Set. By abuse of notation, we also write A for the endofunctor of Set^d given by applying A to each factor:

$$\mathbf{X} \mapsto A\mathbf{X} = (AX_1, \ldots, AX_d)$$

We now proceed to define a generalized sequence natural transformation:

$$\mathsf{seq}_{\mathbf{X}} : \eta(F; A\mathbf{X}) \to A\eta(F; \mathbf{X})$$

(natural in $\mathbf{X}$) of functors $\mathsf{Set}^d \to \mathsf{Set}$.

B.3. **Steps in construction.** Here is a sketch of the steps in defining this natural transformation:

- Recall Assumption 1. Make the following abbreviations of the initial algebras:

  $$\eta(\mathbf{X}) := \eta(F; X_1, \ldots, X_n) \qquad \eta(A\mathbf{X}) := \eta(F; AX_1, \ldots, AX_n).$$

  Initial algebras are fixed points, so we have isomorphisms

  $$c_{\mathbf{X}} : F_{\mathbf{X}}(\eta(\mathbf{X})) \xrightarrow{\sim} \eta(\mathbf{X}) \qquad c_{A\mathbf{X}} : F_{A\mathbf{X}}(\eta(A\mathbf{X})) \xrightarrow{\sim} \eta(A\mathbf{X})$$

- Recall Assumption 2. The natural transformation α and the associativity axiom allow one to define a map[4] on each cofactor:

  $$(AS)^{m_0} \times (AX_1)^{m_1} \times \cdots \times (X_d)^{m_d} \to A(S^{m_0} \times X_1^{m_1} \times \cdots \times X_d^{m_d})$$

  Moreover, the universal property of coproducts implies that there is a map $AX + AY \to A(X + Y)$ (this is true for any endofunctor). We conclude that there is a natural transformation

  $$\kappa_S : F_{A\mathbf{X}}(A \circ S) \to A \circ F_{\mathbf{X}}(S)$$

  (natural in S) of endofunctors of Set.

- The composition

  $$F_{A\mathbf{X}}(A\eta(\mathbf{X})) \xrightarrow{\kappa_{\eta(\mathbf{X})}} AF_{\mathbf{X}}(\eta(\mathbf{X})) \xrightarrow{A(c_{\mathbf{X}})} A\eta(X)$$

[4]This can be done by induction in a straightforward fashion; we omit the details.

is precisely an $F_{A\mathbf{X}}$-algebra structure on $A\eta(\mathbf{X})$. The initiality of $\eta(A\mathbf{X})$ now provides a map (often referred to as a catamorphism):

$$\eta(A\mathbf{X}) \to A\eta(\mathbf{X})$$

which is what we take as the generalized sequence map.

Thus, we have a commutative diagram:

$$\begin{array}{ccccc} F_{A\mathbf{X}}(\eta(A\mathbf{X})) & \xrightarrow{F_{A\mathbf{X}}(\mathsf{seq}_{\mathbf{X}})} & F_{A\mathbf{X}}(A\eta(\mathbf{X})) & \xrightarrow{\kappa_{\eta(\mathbf{X})}} & AF_{\mathbf{X}}(\eta(\mathbf{X})) \\ \downarrow{\scriptstyle c_{A\mathbf{X}}} & & & & \downarrow{\scriptstyle A(c_{\mathbf{X}})} \\ \eta(A\mathbf{X}) & & \xrightarrow{\qquad\qquad\mathrm{seq}_{\mathbf{X}}\qquad\qquad} & & A\eta(\mathbf{X}) \end{array}$$

where the vertical maps are isomorphisms.

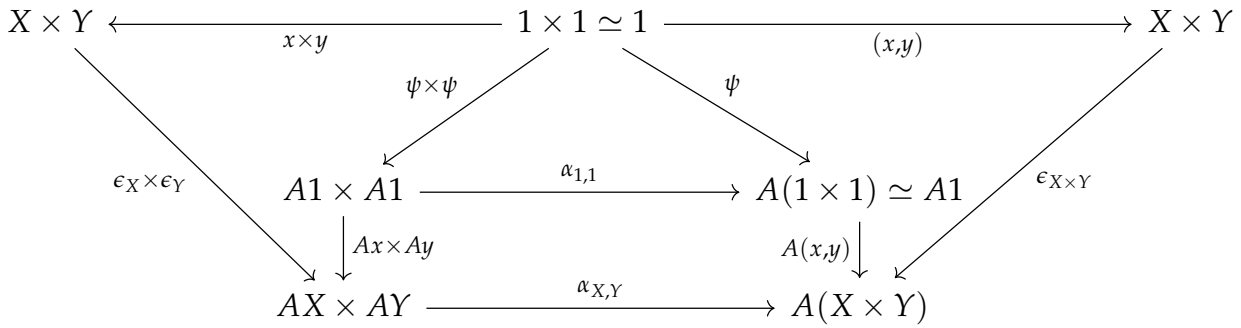
B.4. **Examples.** We invite the reader to revisit each of the examples in Section 6 and comute the sequence map using the binary tree endofunctor BTree instead of the list endofunctor. In the case of the representable monad $\mathrm{Hom}(E, -)$, the sequence map

$$\mathsf{BTree}(\mathrm{Hom}(E, X)) \to \mathrm{Hom}(E, \mathsf{BTree}(X))$$

takes a binary tree of functions $E \to X$ to the function that takes $e \in E$ to the binary tree of elements of X formed by evaluating each function at e.

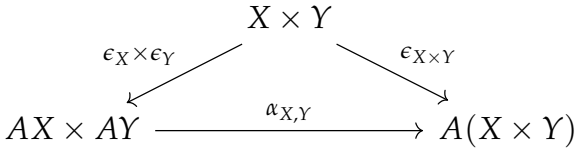
Appendix C. Epsilon

Let (A, ψ, α) be a lax monoidal functor, and let X and Y be sets. We claim the following diagram commutes for any $x \in X$ and $y \in Y$:

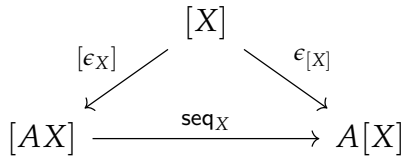


where, by abuse of notation, $x : 1 \to X$, $y : 1 \to Y$, and $(x, y) : 1 \to X \times Y$ are the maps that pick out the indicated points. The commutativity of the triangle in the middle follows from the unit axiom. The commutativity of the bottom square commutes follows from the naturality of α. The commutativity of the outer diagrams on the right and the left follow from the definition of ϵ. Consequently, the following diagram commutes:

(C.1)



We claim that the following diagram commutes as well:



We proceed by induction using the isomorphism $\langle \mathsf{nil}_X, \mathsf{cons}_X \rangle : 1 + X \times [X] \to [X]$. For the empty list, we have:

$$\mathsf{seq}_X \circ [\epsilon_X](\mathsf{nil}_X(1)) = \mathsf{seq}_X(\mathsf{nil}_{AX}(1)) = \epsilon_{[X]}(\mathsf{nil}_X(1))$$

Given a non-empty list $\mathsf{cons}_X(x, \mathbf{x})$, we have:

$$\begin{aligned}
\mathsf{seq}_X \circ [\epsilon_X](\mathsf{cons}_X(x, \mathbf{x})) &= \mathsf{seq}_X(\mathsf{cons}_{AX}(\epsilon_X(x), [\epsilon_X](\mathbf{x}) \\
&= A(\mathsf{cons}_X) \circ \alpha_{X,[X]}(\epsilon_X(x), \mathsf{seq}_X([\epsilon_X](\mathbf{x})) \\
&= A(\mathsf{cons}_X) \circ \alpha_{X,[X]}(\epsilon_X(x), \epsilon_{[X]}(\mathbf{x})) \\
&= A(\mathsf{cons}_X) \circ \alpha_{X,[X]} \circ (\epsilon_X \times \epsilon_{[X]})(x, \mathbf{x}) \\
&= A(\mathsf{cons}_X) \circ \epsilon_{X \times [X]}(x, \mathbf{x}) \\
&= \epsilon_{[X]} \circ \mathsf{cons}_X(x, \mathbf{x})
\end{aligned}$$

where the first equality follows from the definition of $[\epsilon_X]$, the second from the definition of seq_X, the third from induction hypothesis, the fourth from a basic rearrangement, the fifth from the commutativity of the diagram in Diagram C.1, and the sixth from the naturality of ϵ.