0.1 Definition 1–Using natural transformations

Let $\mathcal{C}$ be a category.

A monad on $\mathcal{C}$ consists of:

- a functor $T : \mathcal{C} \to \mathcal{C}$ (endofunctor)
- a natural transformation η called the unit:

$$\begin{tikzcd} \mathcal{C} \arrow[r, "id"] \arrow[d, phantom, "\eta\Downarrow"] & \mathcal{C} \\ \mathcal{C} \arrow[r, "T"] & \mathcal{C} \end{tikzcd}$$

- a natural transformation μ called multiplication:

$$\begin{tikzcd} \mathcal{C} \arrow[r, "T"] & \mathcal{C} \arrow[r, "T"] \arrow[d, phantom, "\mu\Downarrow"] & \mathcal{C} \\ \mathcal{C} \arrow[rr, "T"] & & \mathcal{C} \end{tikzcd}$$

satisfying three properties:

- the right identity law

$$\begin{tikzcd} T \arrow[r, "\eta T"] \arrow[dr, "id"'] & T^2 \arrow[d, "\mu"] \\ & T \end{tikzcd}$$

- the left identity law

$$\begin{tikzcd} T \arrow[r, "T\eta"] \arrow[dr, "id"'] & T^2 \arrow[d, "\mu"] \\ & T \end{tikzcd}$$

- the associativity law

$$\begin{tikzcd} T^3 \arrow[r, "\mu T"] \arrow[d, "T\mu"'] & T^2 \arrow[d, "\mu"] \\ T^2 \arrow[r, "\mu"'] & T \end{tikzcd}$$

It is similar to a monoid, so the name is not a coincidence:

- set X
- element $e \in X$
- function $* : X^2 \to X$

Also satisfying three equations:

- $\forall x \in X. x * e = x$
- $\forall x \in X. e * x = x$
- $\forall x, y, z \in X. (x * y) * z = x * (y * z)$

0.2 Definition 2–As a Kleisli triple

Let $\mathcal{C}$ be a category.

A Kleisli triple on $\mathcal{C}$ consists of:

- for each object A, we have an object TA and a morphism $\eta_A : A \to TA$.
- for objects A, B and morphism $f : A \to TB$, we have a morphism $f* : TA \to TB$.

such that the following equations are satisfied:

- left identity law:

  $$\begin{tikzcd} A \arrow[rr, "f"] \arrow[dr, "\eta"'] & & TB \\ & TA \arrow[ur, "f*_{A,B}"'] & \end{tikzcd}$$

  analogous to:

  $a : A, f : A \to TB \vdash$ `return` $a >>= f = f\ a$

  in Haskell.

- right identity law: $\quad \eta_B *_{B,B} = id_{TB}$

  analogous to:

  $p : TB \vdash p >>= \lambda x.$ `return` $x = p : TB$

  in Haskell.

- associativity law:

$$\begin{tikzcd} TA \arrow[r, "f*"] \arrow[dr, "(f;g)*"'] & TB \arrow[d, "g*"] \\ & TC \end{tikzcd}$$

  analogous to:

  $p : TA, f : A \to TB, g : B \to TC$
  $\vdash (p >>= f) >>= g = p >>= (f >>= g)$
  in Haskell.

Even though the Kleisli triple makes no mention of naturality, it is equivalent to the previous definition given.

0.3 Example: Maybe and Exception monad

$Maybe\ X = X + 1 = \{inlx | x \in X\} \cup \{inr()\}$

$Exc_E\ X = X + E$ where E is a set of behaviours.

The Exc (Exception monad) generalises Id (Identity monad) and $Maybe$ monad:

- Id: $E = \ = \varnothing$
- $Maybe$: $E = 1 = \{()\}$

In **Set**, where morphisms are functions, let's check that Exc_E is indeed a monad.

1. First we map functions to Exc_E:

$$\begin{tikzcd} X \arrow[d, "f"'] & X + E \arrow[d] & inl\ x \arrow[d, mapsto] & inr\ e \arrow[d, mapsto] \\ Y \arrow[r, phantom, "\longmapsto"] & Y + E & inl\ fx & inr\ e \end{tikzcd}$$

2. Unit at X (the return) is

$$\begin{tikzcd} X \arrow[r] & X + E \\ x \arrow[r, mapsto] & inl\ x \end{tikzcd}$$

   We can check that this is natural (as in definition 1):

$$\text{for} \quad \begin{tikzcd} X \arrow[d, "f"'] \\ Y \end{tikzcd} \qquad \begin{tikzcd} X \arrow[r, "inl"] \arrow[d, "f"'] & X + E \arrow[d, "Tf"] \\ Y \arrow[r, "inl"] & Y + E \end{tikzcd}$$

therefore we have a single case

$$\begin{tikzcd} x \arrow[r, mapsto, "inl"] \arrow[d, mapsto, "f"'] & inl\ x \arrow[d, mapsto, "Tf"] \\ fx \arrow[r, mapsto, "inl"] & inl\ f\ x \end{tikzcd}$$ which does commute

so $$\begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[d, "f"'] & TX \arrow[d, "Tf"] \\ Y \arrow[r, "\eta_Y"] & TY \end{tikzcd}$$

3. Multiplication $(T^2X \xrightarrow{\mu_X} TX)$ μ is:

   $\mu_X : (X+E)+E \to X+E$

   $$\begin{aligned} inl\ inl\ x &\longmapsto inl\ x \\ inl\ inr\ e &\longmapsto inr\ e \\ inr\ e &\longmapsto inr\ e \end{aligned}$$

   Once again, we can check it is natural:

   for $$\begin{tikzcd} X \arrow[d, "f"'] \\ Y \end{tikzcd} \qquad \begin{tikzcd} (X+E)+E \arrow[r, "\mu_X"] \arrow[d, "(f+E)+E"'] & X+E \arrow[d, "f+E"] \\ (Y+E)+E \arrow[r, "\mu_Y"] & Y+E \end{tikzcd}$$

   we have to check 3 cases:

   (a) $$\begin{tikzcd} inl\ inl\ x \arrow[r, mapsto, "\mu_x"] \arrow[d, mapsto, "f"'] & inl\ x \arrow[d, mapsto, "f"] \\ inl\ inl\ fx \arrow[r, mapsto, "\mu_x"] & inl\ fx \end{tikzcd}$$

   (b) $$\begin{tikzcd} inl\ inr\ e \arrow[r, mapsto] \arrow[d, mapsto] & inr\ e \arrow[d, mapsto] \\ inl\ inr\ e \arrow[r, mapsto] & inr\ e \end{tikzcd}$$

   (c) $$\begin{tikzcd} inr\ e \arrow[r, mapsto] \arrow[d, mapsto] & inr\ e \arrow[d, mapsto] \\ inr\ e \arrow[r, mapsto] & inr\ e \end{tikzcd}$$

   which all commute.

4. Finally, we have to check the three properties:

   (a) left identity law:

$$\begin{tikzcd} T \arrow[r, "\eta T"] \arrow[dr, "id"'] & T^2 \arrow[d, "\mu"] \\ & T \end{tikzcd}$$

i.e. $\quad \forall X \in C$

$$\begin{tikzcd} TX \arrow[r, "(\eta T)_X"] \arrow[dr, "id_X"'] & T^2X \arrow[d, "\mu"] \\ & TX \end{tikzcd}$$

This can be defined using left-whiskering:

$$\begin{tikzcd} B \arrow[r, "H"] & C \arrow[r, "F"] & D \arrow[d, phantom, "\Downarrow\alpha"] \\ B \arrow[r, "H"] & C \arrow[r, "G"] & D \end{tikzcd} \quad \text{therefore} \quad X \longmapsto \begin{tikzcd} FHX \arrow[d, "\alpha_{HX}"] \\ GHX \end{tikzcd}$$

$\therefore$

$$\begin{tikzcd} TX \arrow[r, "\eta_{TX}"] \arrow[dr, "id_X"'] & T^2X \arrow[d, "\mu"] \\ & TX \end{tikzcd}$$

   (b) right identity law:

$$\begin{tikzcd} TX \arrow[r, "(T\eta)_X"] \arrow[dr, "id_X"'] & T^2X \arrow[d, "\mu"] \\ & TX \end{tikzcd}$$

This can be defined using right-whiskering:

$$\begin{tikzcd} C \arrow[r, "F"] \arrow[d, phantom, "\Downarrow\alpha"] & D \arrow[r, "K"] & E \\ C \arrow[r, "G"] & D \arrow[r, "K"] & E \end{tikzcd} \quad \text{therefore} \quad X \longmapsto \begin{tikzcd} KFX \arrow[d, "K\alpha_X"] \\ KGX \end{tikzcd}$$

$\therefore$

$$\begin{tikzcd} TX \arrow[r, "T\eta_X"] \arrow[dr, "id_X"'] & T^2X \arrow[d, "\mu"] \\ & TX \end{tikzcd}$$

   (c) associativity law:

$$\begin{tikzcd}
T^3X \arrow[r, "\mu_{TX}"] \arrow[d, "T\mu_X"'] & T^2X \arrow[d, "\mu_X"] \\
T^2X \arrow[r, "\mu_X"'] & TX
\end{tikzcd}$$

Now the actual proof that the properties indeed hold:

(a) There are two cases to show the left-identity law; *inl x* and *inr e*:

$$\begin{tikzcd}
\mathit{inl}\ x \arrow[r, mapsto, "\mathit{inl}"] \arrow[dr, mapsto, "\mathit{id}"'] & \mathit{inl}\ \mathit{inl}\ x \arrow[d, mapsto, "\mu"] \\
& \mathit{inl}\ x
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
\mathit{inr}\ e \arrow[r, mapsto, "\mathit{inl}"] \arrow[dr, mapsto, "\mathit{id}"'] & \mathit{inl}\ \mathit{inr}\ e \arrow[d, mapsto, "\mu"] \\
& \mathit{inr}\ e
\end{tikzcd}$$

(b) There are two cases to show the right-identity law; *inl x* and *inr e*:

$$\begin{tikzcd}
\mathit{inl}\ x \arrow[r, mapsto, "T\ \mathit{inl}"] \arrow[dr, mapsto, "\mathit{id}"'] & \mathit{inl}(\mathit{inr}\ x) \arrow[d, mapsto, "\mu"] \\
& \mathit{inl}\ x
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
\mathit{inr}\ e \arrow[r, mapsto, "T\ \mathit{inl}"] \arrow[dr, mapsto, "\mathit{id}"'] & \mathit{inl}(\mathit{inr}\ e) \arrow[d, mapsto, "\mu"] \\
& \mathit{inr}\ e
\end{tikzcd}$$

(c) There are four cases to show the associativity law; *inl inl inl x*, *inl inl inr e*, *inl inr e*, and *in e*:

$$\begin{tikzcd}
\mathit{inl}\ \mathit{inl}\ \mathit{inl}\ x \arrow[r, "\mu_{X+E}"] \arrow[d, "\mu_X + E"'] & \mathit{inl}\ \mathit{inl}\ x \arrow[d, "\mu_X"] \\
\mathit{inl}(\mathit{inl}\ x) \arrow[r, "\mu_X"'] & \mathit{inl}\ x
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathit{inl}\ \mathit{inl}\ \mathit{inr}\ e \arrow[r, "\mu_{X+E}"] \arrow[d, "\mu_X + E"'] & \mathit{inl}\ \mathit{inr}\ e \arrow[d, "\mu_X"] \\
\mathit{inl}\ \mathit{inl}\ e \arrow[r, "\mu_X"'] & \mathit{inr}\ e
\end{tikzcd}$$

$$\begin{tikzcd}
\mathit{inl}\ \mathit{inl}\ e \arrow[r, "\mu_{X+E}"] \arrow[d, "\mu_X + E"'] & \mathit{inr}\ e \arrow[d, "\mu_X"] \\
\mathit{inl}\ \mathit{inr}\ e \arrow[r, "\mu_X"'] & \mathit{inr}\ e
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathit{inr}\ e \arrow[r, "\mu_{X+E}"] \arrow[d, "\mu_X + E"'] & \mathit{inr}\ e \arrow[d, "\mu_X"] \\
\mathit{inr}\ e \arrow[r, "\mu_X"'] & \mathit{inr}\ e
\end{tikzcd}$$

With this, we have shown that the properties hold, and that Exc_E is indeed a monad.

0.4 Kleisli extension

Monads are many times implemented using the Kleisli extension. As with Haskell.

$(-)^* : hom(X, TY) \to hom(TX, TY)$

Given a monad $< T, \eta, \mu >$ over category C and a morphism $f : X \to TY$:

$f^* : TX \to TY$
$f^* = \mu_Y \circ Tf$

In other words:

$f : X \to TY$ is any function that gives us the monad type in Haskell. This also matches η, which is equivalent to the `return` function.

$f^* : TX \to TY$ is analogous to the bind function in Haskell. The difference is that arguments are in different order.

For example, if we apply this to the Exc_E monad:

$X \xrightarrow{f} Y + E$
$X + E \xrightarrow{f^*} Y + E$

so

$inl\ x \mapsto f\ x$
$inr\ e \mapsto inr\ e$

0.5 Comparison with Haskell

Monads can be defined in the standard mathematical way or through the Kleisli triple. We can either have:

- $< T, \eta, \mu >$, where T gives us the type of the monad, η is the `return` and μ is the `join`. Note that T is a functor, so in Haskell, this would be a pair $(T, fmap)$, which maps objects and functions.
- $< T, \eta, (-)^* >$, where T gives us the type, again a pair with `fmap`, η is the `return`, and $(-)^*$ is equivalent to `bind` (»=).

This can be seen more clearly with the types:

`join` and μ (multiplication):

```
join ::  T T A -> T A
```

$\mu : TTA \to TA$

`bind` and $(-)^*$ (Kleisli extension operator):

```
»= ::  T A -> (A -> T B) -> T B
```

$(-)^* : (A \to TB) \to (TA \to TB)$

We can show that a Kleisli triple on C is equivalent to the standard mathematical definition of a monad on C because:

- μ and η give you »=, i.e. $(-)^*$
- $(-)^*$ and η give you `join`, i.e. μ

0.6 Exercise: State monad

Let S be a set (for some state, i.e. the type of the state/store).

We can define a monad on **Set** with $T\ A = S \to (S \times A)$

In programming, $T\ A$ would be a stateful program that returns a value of type A.

For instance, if $A = \mathbb{N}$, then $T\ \mathbb{N} = S \to (S \times \mathbb{N})$.

We could write a program that uses this state:

$\lambda b : \texttt{bool}.(\texttt{not}\ b, b) : T\ \texttt{bool}$

This program would take the original state, invert it, and then output the original state.

With that context, we can define the monad either the standard way, or as a Kleisli triple. Defining it as a Kleisli triple:

Let *State* be the monad:

- $State\ X = S \to (S \times X)$
- The Unit (η)
  $\eta : X \to (S \to (S \times X))$
  $\eta = x \mapsto (s \mapsto (s, x))$ where $s : S$ and $x : X$
- The Kleisli operator (equivalent to bind):
  Given $g : X \to (S \to (S \times Y))$
  $g^* : (S \to (S \times X)) \to (S \to (S \times Y))$
  $g* = f \mapsto s \mapsto (g(\pi_2\ f\ s))\ (\pi_1\ f\ s)$

This is analogous to the Haskell state monad. Given *State* is analogous to our functor T:

```
newtype State s a = State  runState ::  s -> (a, s)

return ::  a -> State s a
return x = State ( \s -> (x, s) )

(»=) ::  State s a -> (a -> State s b) -> State s b
(State h) »= f
= State ( \s -> let (a,new_sate) = h s in f a new_state )
```

Alternatively:

```
type State s a = s → (a,s)
return x = λ s → (x,s)
f »= g   = λ s → case f s of (x,s') → g x s'
```