Monads: From Functional Programming to Logic

Haitian Wang

2 March 2026

Outline

Overview

Monoids vs Categories

Monads: a programming view

Monads: an adjunctions view

Monads: an algebraic view

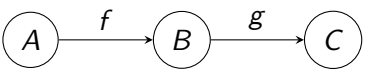
Bonus: Applicative Functors

Monoids vs Categories

Monoid

untyped language

Category



typed language

Overview

Monoids vs Categories

Monads: a programming view

Monads: an adjunctions view

Monads: an algebraic view

Bonus: Applicative Functors

Monads as bind and return

```
class Applicative m => Monad (m :: * -> *) where
    (>>=) :: m a -> (a -> m b) -> m b
    (>>) :: m a -> m b -> m b
    return :: a -> m a
    {-# MINIMAL (>>=) #-}
```

satisfying:

```
    return a >>= k                  =  k a              -- left identity (law 1)
    m        >>= return             =  m                -- right identity (law 2)
    m        >>= (\x -> k x >>= h)  =  (m >>= k) >>= h  -- associative (law 3)
```

But why bind and return and why these laws?

Monads as fish and eta(return)

Instead of bind and return, we can have fish and eta:
eta + fish = eta(return) + bind = fmap + eta + mu

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

satisfying:

```
    eta >=> f = f                          -- left unit
    f >=> eta = f                          -- right unit
    (f >=> g) >=> h = f >=> (g >=> h) -- associativity
```

which are the composition and identity (laws) in the Kleisli Category.

An informal definition of Kleisli category: In a Kleisli category (w.r.t an endofunctor M), the objects are the same but the arrows from A to B are defined to be arrows from A to MB in the original category:
$Hom_{\mathcal{K}_M}(A, B) := Hom_{\mathcal{C}}(A, MB)$.

Overview

Monoids vs Categories

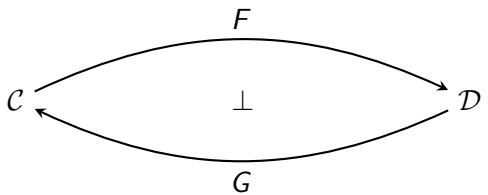
Monads: a programming view

Monads: an adjunctions view

Monads: an algebraic view

Bonus: Applicative Functors

Adjunctions



$$\mathrm{Hom}_{\mathcal{D}}(FC, D) \cong \mathrm{Hom}_{\mathcal{C}}(C, GD)$$

Adjunctions as some level of similarity

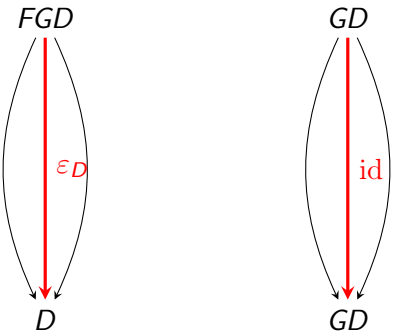
Five levels of similarity between categories from weak to strong:

- Existence of functors (F, G)
- Existence of adjunctions $(F \dashv G)$
- Categorically equivalent $(C \simeq D)$
- Categorically isomorphic $(C \cong D)$
- Identical $(C = C)$

Unit and counit

$$\mathrm{Hom}_{\mathcal{D}}(FC, D) \cong \mathrm{Hom}_{\mathcal{C}}(C, GD)$$

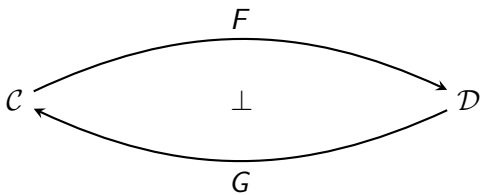
If we let C be GD:



For each D, we have an arrow $\epsilon_D : FGD \to D$. We actually have a natural transformation $\epsilon : FG \Rightarrow 1_{\mathcal{D}}$, called the counit of the adjunction. Similarly, we can let D be FC and have unit $\eta : 1_{\mathcal{C}} \Rightarrow GF$.

Monad arising from adjunctions

Given an adjunction (F, G, ϵ, η):

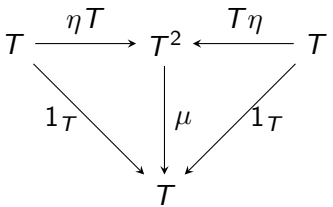


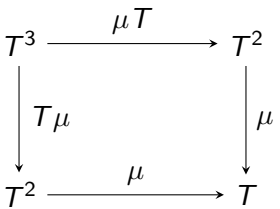
We have

- an endofunctor $T = GF : \mathcal{C} \to \mathcal{C}$
- a natural transformation (the unit) $\eta : 1_{\mathcal{C}} \Rightarrow T$
- another natural transformation $\mu : T^2 \Rightarrow T$ where $\mu_{\mathcal{C}} := G(\epsilon_{FC}) : T^2 C \to TC$

Monad Laws

Given (T, η, μ), the following diagrams commute:

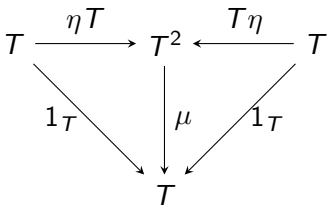


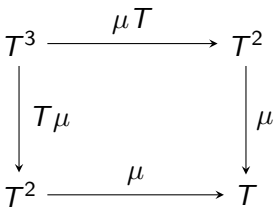


We define a monad to be a triple (T, η, μ) satisfying these identities. So every pair of adunctions $F \dashv G$ gives rise to a monad $T = GF$. Dually, we also have a comonad $(L = FG, \epsilon : L \Rightarrow 1_{\mathcal{D}}, \delta = F\eta G : L \Rightarrow L^2)$.

Monad Laws

Given (T, η, μ), the following diagrams commute:

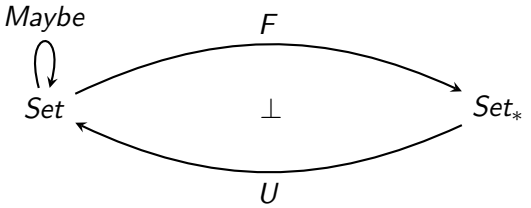




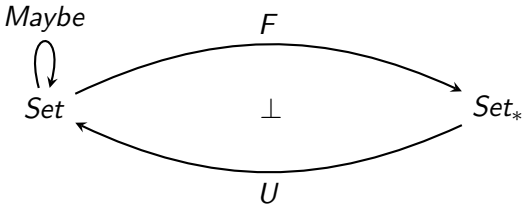
We define a monad to be a triple (T, η, μ) satisfying these identities. So every pair of adunctions $F \dashv G$ gives rise to a monad $T = GF$. Dually, we also have a comonad $(L = FG, \epsilon : L \Rightarrow 1_{\mathcal{D}}, \delta = F\eta G : L \Rightarrow L^2)$.

So every pair of adjunctions gives a (co)monad. Conversely, every monad arises from some pair of adjunctions but in more than one way.

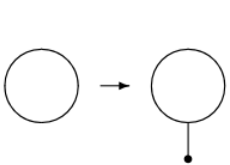
Example: Maybe



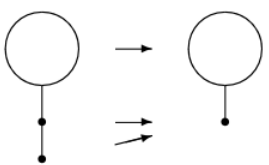
Example: Maybe



The monad $Maybe = UF$ with unit and multiplication:

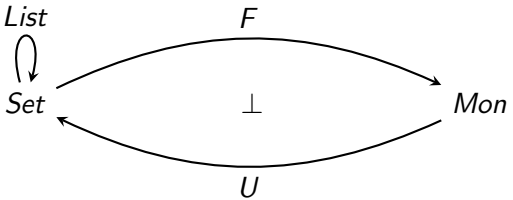


$\eta_X : X \to T(X)$

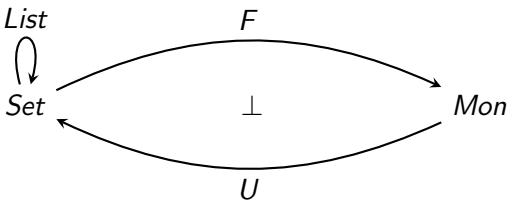


$\mu_X : T^2(X) \to T(X)$

Example: Lists

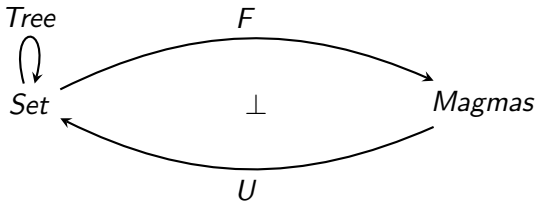


Example: Lists

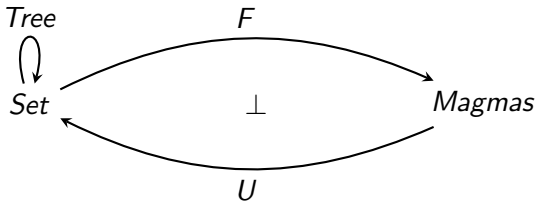


Given a set Σ (think of it as a set of alphabets), F gives the free monoid $(M, \cdot)$ arising from the set Σ, where M contains all the finite strings and the multiplication $\cdot$ is just $++$.

Example: (Binary) Trees

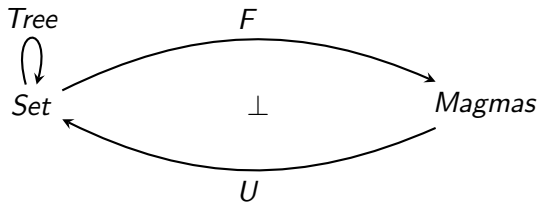


Example: (Binary) Trees



Given a set Σ (think of it as a set of alphabets), F gives the free magmas $(M, \cdot)$ arising from the set Σ, where M contains all the syntax trees and the operation $\cdot$ is the action of joining trees at the root.

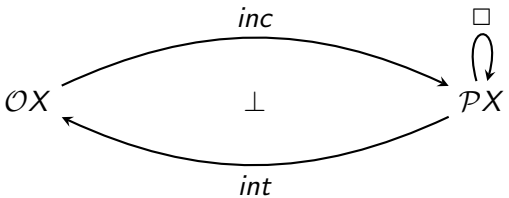
Example: (Binary) Trees



Given a set Σ (think of it as a set of alphabets), F gives the free magmas $(M, \cdot)$ arising from the set Σ, where M contains all the syntax trees and the operation $\cdot$ is the action of joining trees at the root.

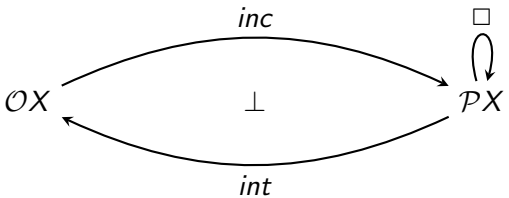
The list is a special case of binary trees when we have the equations $e \cdot x = x \cdot e = x$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

An example of comonad: The $\Box$ in S4



Given a topological space $(X, \mathcal{O}X)$, $int(S)$ is the largest open subset of $S \subseteq X$ and inc is just the inclusion map.

An example of comonad: The $\Box$ in S4

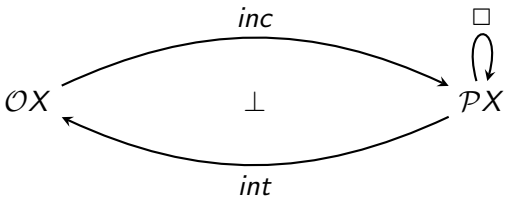


Given a topological space $(X, \mathcal{O}X)$, $int(S)$ is the largest open subset of $S \subseteq X$ and inc is just the inclusion map.

Recall the axioms (other than K) for S4:

- T: $\Box p \to p$
- 4: $\Box p \to \Box\Box p$

An example of comonad: The $\Box$ in S4



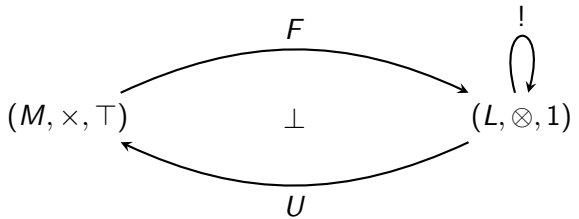
Given a topological space $(X, \mathcal{O}X)$, $int(S)$ is the largest open subset of $S \subseteq X$ and inc is just the inclusion map.

Recall the axioms (other than K) for S4:

- T: $\Box p \to p$
- 4: $\Box p \to \Box\Box p$

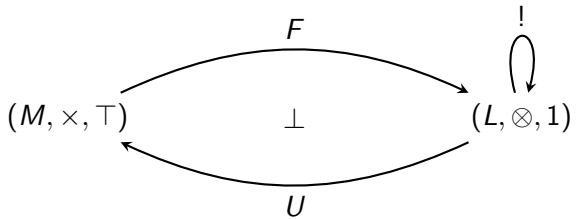
The adjunction shows that intuitionistic logic is the 'interior-stable' fragment of S4. Or with an epistemic reading, intuitionitic logic is the logic of verifiable propositions.

Example: The !-modality in Linear Logic



M is cartesian monoidal and L is symmetric monoidal.

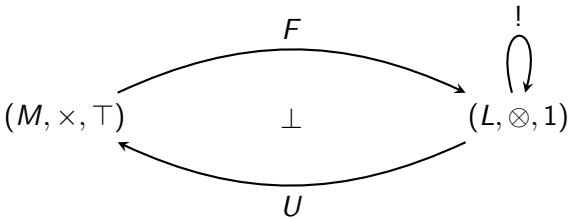
Example: The !-modality in Linear Logic



M is cartesian monoidal and L is symmetric monoidal.

U forgets the usage restriction while FA gives the duplicable counterpart of A in the linear world.

Example: The !-modality in Linear Logic

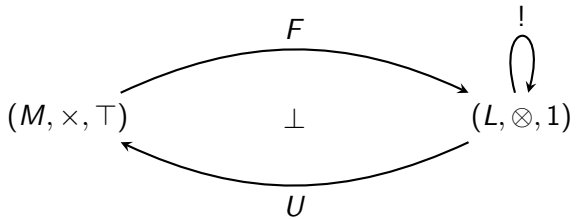


M is cartesian monoidal and L is symmetric monoidal.

U forgets the usage restriction while FA gives the duplicable counterpart of A in the linear world.

The $Hom_M(A, UB) \cong Hom_L(FA, B)$ can be interpreted as $A \Rightarrow B \cong !A \multimap B$.

Example: The !-modality in Linear Logic



M is cartesian monoidal and L is symmetric monoidal.

U forgets the usage restriction while FA gives the duplicable counterpart of A in the linear world.

The $Hom_M(A, UB) \cong Hom_L(FA, B)$ can be interpreted as $A \Rightarrow B \cong !A \multimap B$.

This shows that we can embed intuitionistic logic into linear logic as the fragment restricted to persistent objects.

Overview

Monad as a monoid

Monad is a monoid in the category of endofunctors.

Monad as a monoid

Monad is a monoid in the category of endofunctors.

The category of endofunctors forms a monoidal category: $([\mathcal{C}, \mathcal{C}], \circ, Id_{\mathcal{C}})$.

Monad as a monoid

Monad is a monoid in the category of endofunctors.

The category of endofunctors forms a monoidal category: $([\mathcal{C}, \mathcal{C}], \circ, Id_{\mathcal{C}})$.

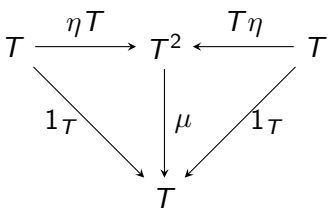
A monoid object in the monoidal category is precisely
$(T, \mu : T^2 \to T, \eta : Id_{\mathcal{C}} \to T)$ satisfying the associativity and unit laws:

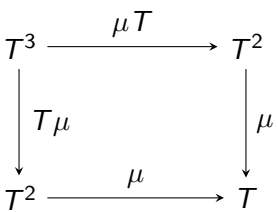
Monad as a monoid

Monad is a monoid in the category of endofunctors.

The category of endofunctors forms a monoidal category: $([\mathcal{C},\mathcal{C}], \circ, Id_{\mathcal{C}})$.

A monoid object in the monoidal category is precisely $(T, \mu : T^2 \to T, \eta : Id_{\mathcal{C}} \to T)$ satisfying the associativity and unit laws:





which are exactly the monad laws we see earlier.

Overview

Monoids vs Categories

Monads: a programming view

Monads: an adjunctions view

Monads: an algebraic view

Bonus: Applicative Functors

Applicative functors as lax closed/monoidal functors

The familiar Haskell definition of applicative functor:

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

The definition of lax closed functor:

```
class Functor f => Closed f where
    unit :: f ()
    (<*>) :: f (a -> b) -> f a -> f b
```

The definition of lax monoidal functor:

```
class Functor f => Monoidal f where
    unit :: f ()
    (>*<) :: (f a, f b) -> f (a, b)
```

The three definitions are equivalent in Haskell since Hask is assumed to be a closed symmetric monoidal category.

Applicative functors as lax closed/monoidal functors

The familiar Haskell definition of applicative functor:

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

The definition of lax closed functor:

```
class Functor f => Closed f where
    unit :: f ()
    (<*>) :: f (a -> b) -> f a -> f b
```

The definition of lax monoidal functor:

```
class Functor f => Monoidal f where
    unit :: f ()
    (>*<) :: (f a, f b) -> f (a, b)
```

The three definitions are equivalent in Haskell since Hask is assumed to be a closed symmetric monoidal category.

The upshot: Applicative functors preserve monoidal structures.

Main References

[1] Milewski, Bartosz. Category theory for programmers. Bartosz Milewski, 2019.

[2] Yukita, Shuichi. "Category Theory Using Haskell." Computer science foundations and applied logic.(2024).

[3] Kishida, Kohei. "Categories and modalities." Categories for the Working Philosopher. Oxford University Press, 2018. 163-222.

[4] van den Berg, Benno. The Art of Composition. Draft Lecture Notes (2025).

More References and Links

- Awodey, Steve. Category theory. Vol. 52. OUP Oxford, 2010.
- Bartosz Milewski's blogpost on Applicative functors (2017). https://bartoszmilewski.com/2017/02/06/applicative-functors/
- stackoverflow's discussion of What is Applicative Functor definition from the category theory POV? https://stackoverflow.com/questions/35013293/what-is-applicative-functor-definition-from-the-category-theory-pov
- nLab pages on !-modality, monoidal category, closed category, tensorical strength, etc. https://ncatlab.org/nlab/show/monoidal+category
- Andrej Bauer's blogpost on Hask is not a category. https://math.andrej.com/2016/08/06/hask-is-not-a-category/
- Julio Song's blogpost on adjunctions. https://blog.juliosong.com/linguistics/mathematics/category-theory-notes-12/
- Alexander Kurz's blogpost on monads. https://hackmd.io/@alexhkurz/ByD5fgecY
- Tsuchiya, Naotsugu, Shigeru Taguchi, and Hayato Saigo. "Using category theory to assess the relationship between consciousness and integrated information theory." Neuroscience research 107 (2016): 1-7.

Some references that I wanted to cover but didn't have time for:

- Andrej Bauer's blog post on Hask is not a category.
  `https://math.andrej.com/2016/08/06/hask-is-not-a-category/`
- More on modality and monads
  - Moggi, Eugenio. "Notions of computation and monads." Information and computation 93.1 (1991): 55-92.
  - Kobayashi, Satoshi. "Monad as modality." Theoretical Computer Science 175.1 (1997): 29-74.