Monads from Scratch

TTU Functional Programming Club
March 21, 2017

Presented by
Silvio Mayolo

Monads from Scratch

- "Definition" of a Monad
  - (As of GHC 7.10)
  -
```
class Applicative m => Monad m where
    return :: a -> m a
    (>>=) :: m a -> (a -> m b) -> m b
```
  -
  - Also relevant:
  -
```
join :: Monad m => m (m a) -> m a
join x = x >>= id
```
  -

Monads from Scratch

- "Definition" of a Monad
  - Haskell's definition: return, (>>=)
  - Traditional definition: fmap, return, join
  - Note: (>>=) can be written in terms of join
  - ```haskell
    (>>=) :: Monad m => m a -> (a -> m b) -> m b
    x >>= f = join $ fmap f x
    ```
  - And vice versa
  - ```haskell
    join :: Monad m => m (m a) -> m a
    join x = x >>= id
    ```

Monads from Scratch

- "Definition" of a Monad
  - Mathematical jargon:

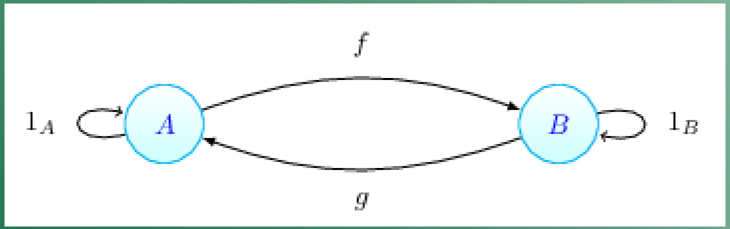
Haskell Name	Mathematical Name	Notation
fmap	functor	F
return	unit	Greek letter eta
join	join	Greek letter mu

Monads from Scratch

- Start from the Bottom – Categories
  - Category is objects & arrows
  - Objects depend on the domain of discourse
  - Arrows are maps between objects

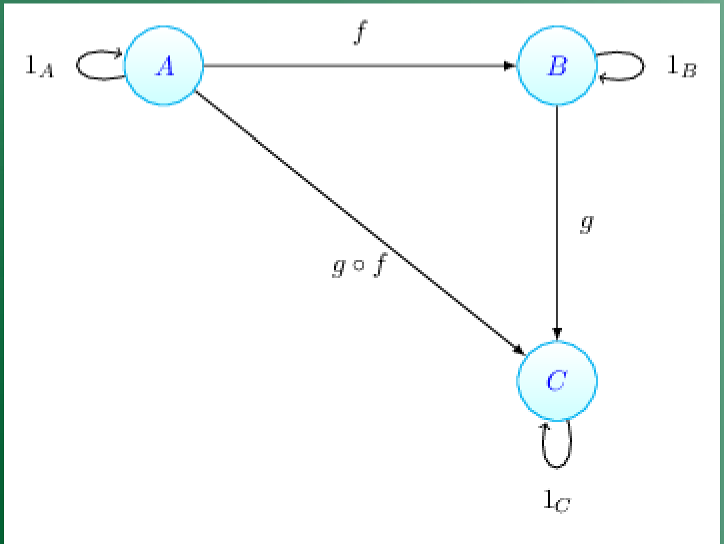
Monads from Scratch

- Start from the Bottom – Categories
  - A simple category
  -
  -
  -



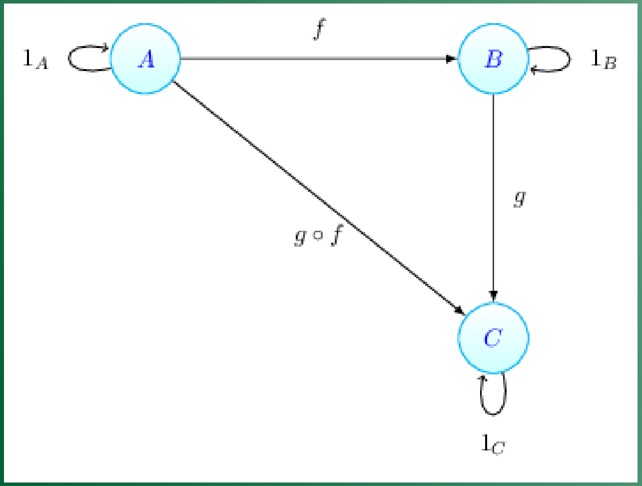
Monads from Scratch

- Start from the Bottom – Categories
  - Every object has an identity arrow
  - Arrows can be “composed” to make new arrows
  -
  -
  -
  -
  -

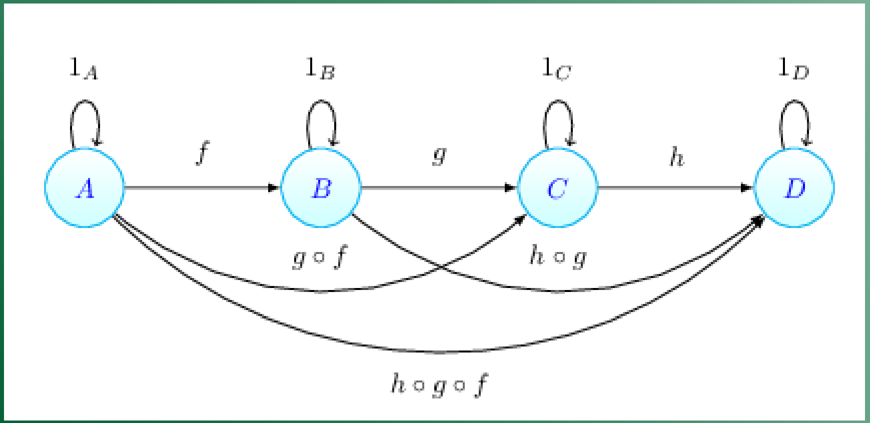


Monads from Scratch

- Start from the Bottom – Categories
  - Unfortunate issue with notation
  - “g of f” vs “f of g”
  - Functional notation vs standard notation



Monads from Scratch

- Start from the Bottom – Categories
  - The Category Laws
  - Identity Law: For $f : A \to B$: $1_B \circ f = f \circ 1_A = f$
    Associative Law: For $f : A \to B, g : B \to C, h : C \to D$: $(h \circ g) \circ f = h \circ (g \circ f)$
  - 

Monads from Scratch

- Start from the Bottom – Categories
  - Commonly used categories
    - The category of sets (**Set**)
    - The category of groups (**Grp**)
    - The category of Haskell types (**Hask**)
    - The category for a partial order

Monads from Scratch

- Start from the Bottom – Categories
  - Is there a category of all categories?
    - What would the objects/arrows be?

Monads from Scratch

- Start from the Bottom – Categories
  - Is there a category of all categories?
    - What would the objects/arrows be?
  - Yes, there is: **Cat**
  - Objects = Categories
  - Arrows = Functors

Monads from Scratch

- Functors – The Arrows of Categories
  - $F : C \to D$
  - Objects in C → Objects in D
  - Arrows in C → Arrows in D

Monads from Scratch

- Functors – The Arrows of Categories
  - F : C → D
  - Objects in C → Objects in D
  - Arrows in C → Arrows in D
  - More rigorously:

A functor $F : \mathcal{C} \to \mathcal{D}$ is defined as an association.

- Every object A in $\mathcal{C}$ is mapped to an object $F(A)$ in $\mathcal{D}$
- Every morphism $f : A \to B$ in $\mathcal{C}$ is mapped to a morphism $F(f) : F(A) \to F(B)$ in $\mathcal{D}$

Monads from Scratch

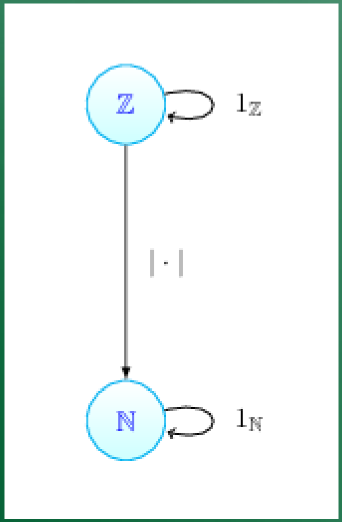
- Functors – The Arrows of Categories
  - Functor Laws
  - Preservation of Identity: For each object A, $F(1_A) = 1_{F(A)}$
    Preservation of Composition: For any f, g, $F(g) \circ F(f) = F(g \circ f)$

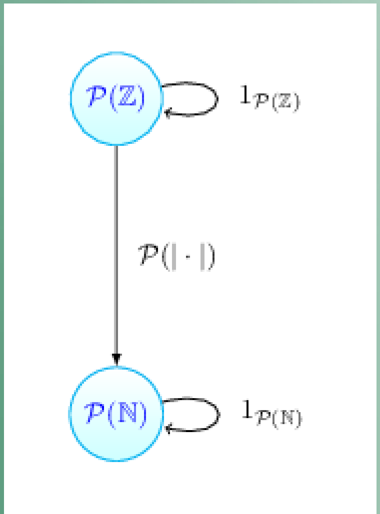
Monads from Scratch

- Functors – The Arrows of Categories
  - Let's look at some examples

Monads from Scratch

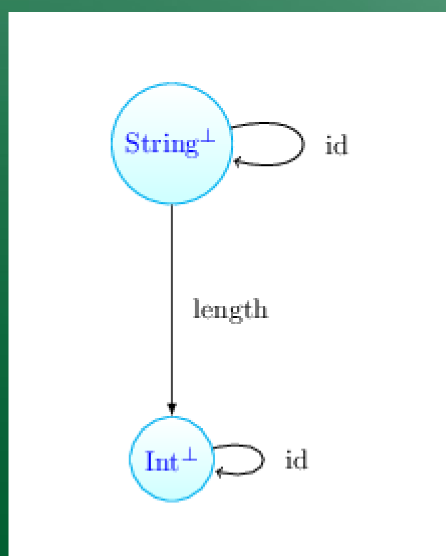
- Functors – The Arrows of Categories
  - Let's look at some examples
  - Power Set Functor (P : **Set** → **Set**)

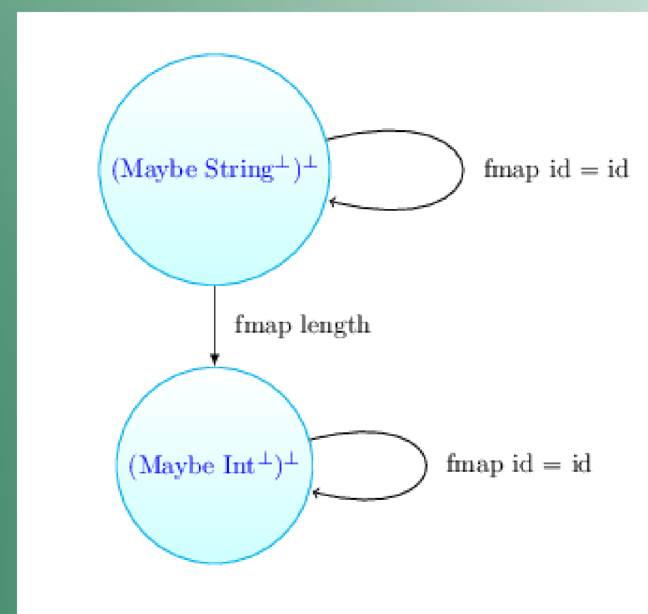




Monads from Scratch

- Functors – The Arrows of Categories
  - Let's look at some examples
  - Maybe Functor (Maybe : **Hask** → **Hask**)
  -
  -
  -
  -
  -





Monads from Scratch

- Functors – The Arrows of Categories
  - Functors allow categories to be “compared”
  - We can now define “isomorphism”

Monads from Scratch

- Functors – The Arrows of Categories
  - What's the next step?
  - We have categories …
  - Arrows between categories are “functors” …
  - Arrows between functors are … ?

Monads from Scratch

- Monads – The Arrows of Functors

Monads from Scratch

- Monads – The Arrows of Functors
  - Nope; we have one more step

Monads from Scratch

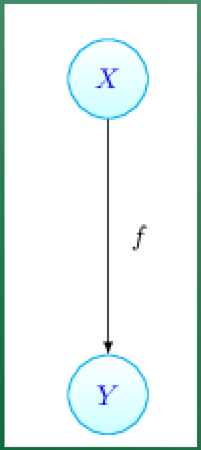
- Natural Transformations – The Arrows of Functors
  - Arrows between functors are natural transformations

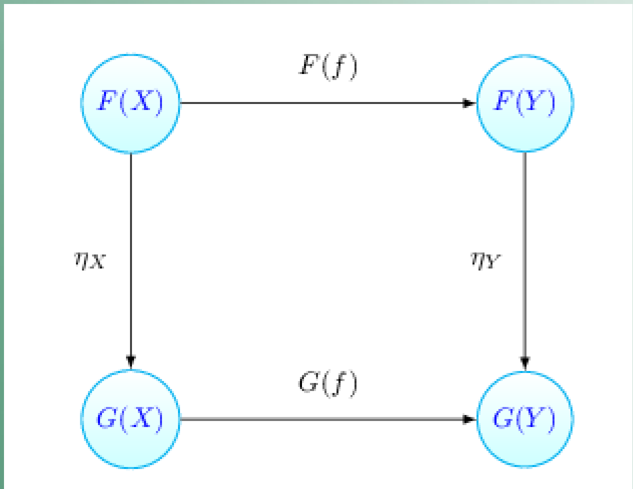
Monads from Scratch

- Natural Transformations – The Arrows of Functors
  - Definition:
  - Given categories $\mathcal{C}$, $\mathcal{D}$ and functors $F, G : \mathcal{C} \to \mathcal{D}$
    A natural transformation: $\eta : F \Rightarrow G$
    Associates each object X in $\mathcal{C}$ with a *morphism* $\eta_X : F(X) \to G(X)$
    Such that $\eta_Y \circ F(f) = G(f) \circ \eta_X$ for any objects X, Y in $\mathcal{C}$
  -

Monads from Scratch

- Natural Transformations – The Arrows of Functors
  - Illustration:





  - This diagram is *commutative*

Monads from Scratch

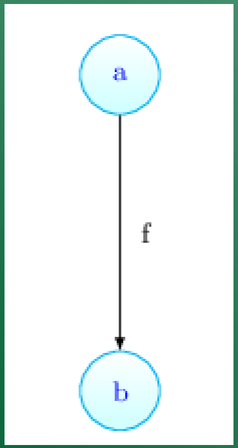
- Natural Transformations – The Arrows of Functors
  - How about a demonstration?
  - In Haskell, “Maybe” is basically just a list with at most one element
  - Can we encode that statement categorically?

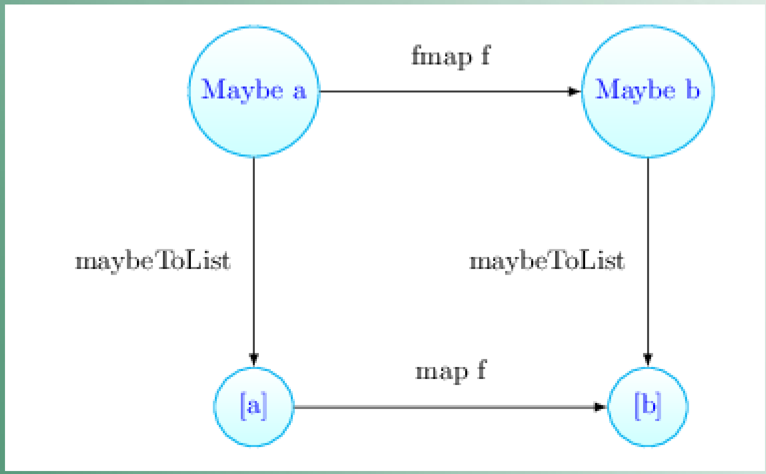
Monads from Scratch

- Natural Transformations – The Arrows of Functors
  - “List” and “Maybe” are already functors
  - What about an arrow between them?
  - From Data.Maybe:
  - ```
    maybeToList :: Maybe a -> [a]
    maybeToList Nothing = []
    maybeToList (Just x) = [x]
    ```
  -
    ```

Monads from Scratch

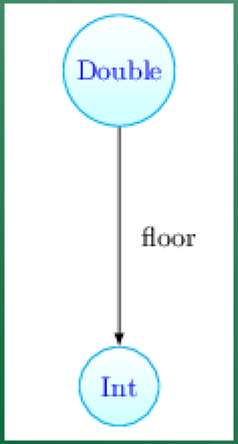
- Natural Transformations – The Arrows of Functors
  - Then, for some arbitrary types *a* and *b* and some function *f*, we have:
  -
  -
  -
  -

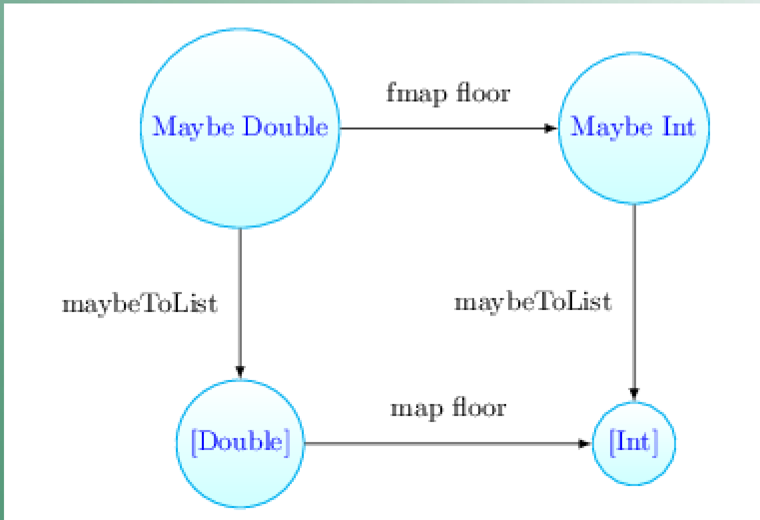




Monads from Scratch

- Natural Transformations – The Arrows of Functors
  - A little more concretely …
  -
  -
  -
  -





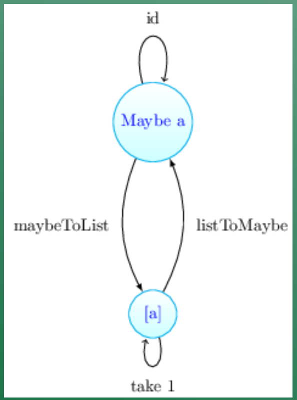
Monads from Scratch

- Natural Transformations – The Arrows of Functors
  - In order for this to be valid, we need the following to be equivalent
  - `maybeToList . fmap floor`   `map floor . maybeToList`
  - So let's ask QuickCheck
  -
```
ghci> quickCheck (\x -> (maybeToList . fmap floor $ x) == (map floor . maybeToList $ x))
+++ OK, passed 100 tests.
```

Monads from Scratch

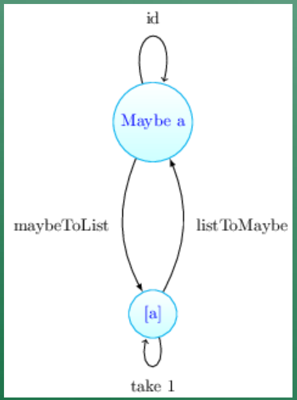
- Natural Transformations – The Arrows of Functors
  - Let's take it further
  - “maybeToList” has a corresponding function “listToMaybe”
  - How do they interact?

Monads from Scratch

- Natural Transformations – The Arrows of Functors
  - 

  -
  -
  -
  - In one direction, inverse
  - In the other direction, information is lost

Monads from Scratch

- Natural Transformations – The Arrows of Functors
  - 

  - This is no accident
  - “List” is, in some sense, more powerful than “Maybe”

Monads from Scratch

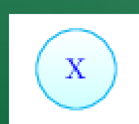
- Monads – The Final Step
  - Now we have all the tools we need

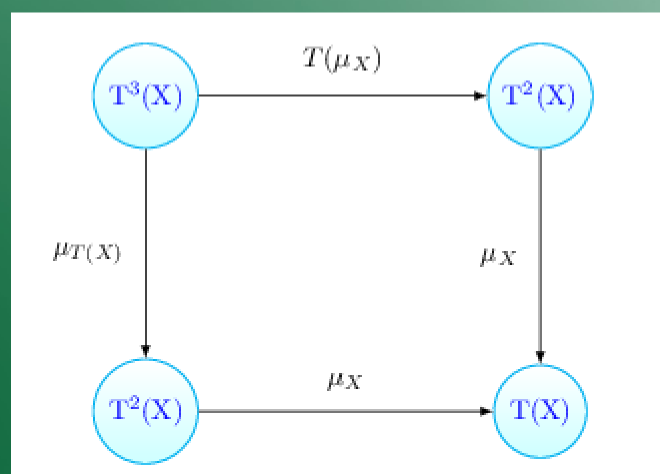
Monads from Scratch

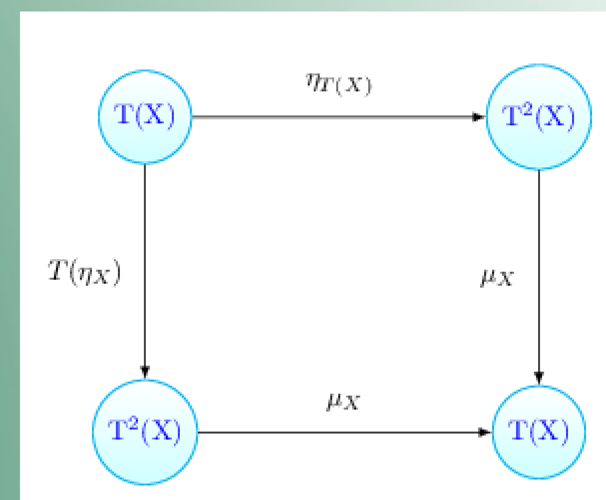
- Monads – The Final Step
  - A monad is a pair of natural transformations (“unit” and “join”) acting on a single endofunctor

Monads from Scratch

- Monads – The Final Step
  - Helpful diagram
  - All of the following commute
  -
  -
  -
  -

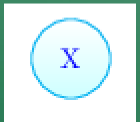


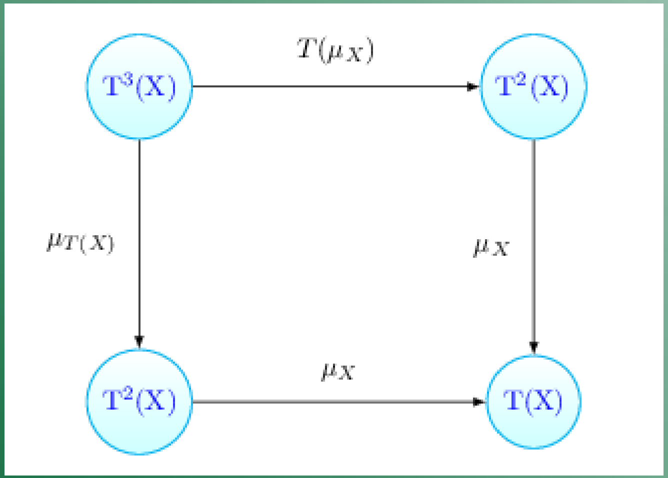


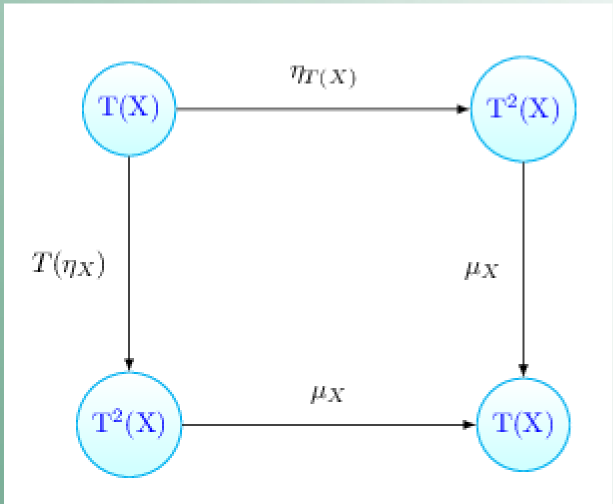


Monads from Scratch

- Monads – The Final Step







  - Demonstration of these laws; pick your favorite monad in Haskell

Monads from Scratch

- Monads – The Final Step
  - We now have all the tools we need to mathematically analyze stateful computations

Monads from Scratch

- Further Reading
  - If interested:
    - The categorical dual gives rise to *comonads*
    - Monads themselves give rise to categories known as *Kleisli categories*
    - A natural transformation which is also an isomorphism is called a *natural isomorphism* and is surprisingly useful in universal algebra

Monads from Scratch

- Further Reading
  - Thank you!