Maybe Monad (3B)

Young Won Lim
12/11/17

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Based on

Haskell in 5 steps
https://wiki.haskell.org/Haskell_in_5_steps

https://www.schoolofhaskell.com/user/EFulmer/currying-and-partial-application

http://www.idryman.org/blog/2014/01/23/yet-another-monad-tutorial/

A Type Monad

Haskell does not have states

It's type system is powerful enough to construct the stateful program flow

defining a Monad type in Haskell
defining a class in an object oriented language (C++, Java)

A Monad can do much more than a class:

A Monad is a type that can be used for

- **exception handling**
- constructing **parallel program workflow**
- a **parser** generator

http://www.idryman.org/blog/2014/01/23/yet-another-monad-tutorial/

Types: rules and data

types are the ***rules*** associated with the **data**,
not the actual **data** itself.

Object-Oriented Programming enable us
- to use classes / interfaces
  - to define **types**,
  - the **rules** (**methods**) that interacts with the actual **data**.

- to use templates(c++) or generics(java)
  - to define more **abstracted rules** that are more reusable

Monad is pretty much like **generic class**.

http://www.idryman.org/blog/2014/01/23/yet-another-monad-tutorial/

Monad Rules

A type is just a set of rules, or methods in Object-Oriented terms

A Monad is just yet another type, and the definition of this type is defined by **four rules**:

1) bind (>>=)
2) then (>>)
3) return
4) fail

http://www.idryman.org/blog/2014/01/23/yet-another-monad-tutorial/

Monad Applications

1. **Exception Handling**

2. **Accumulate States**

3. **IO Monad**

http://www.idryman.org/blog/2014/01/23/yet-another-monad-tutorial/

Monad Class Function >>= & >>

>>= and >> : functions from the Monad *class*

Monad Sequencing Operator with value passing

>>= **passes** the result of the expression on the left
as an argument to the expression on the right,
while preserving the context the argument and function use

Monad Sequencing Operator

>> is used to **order** the **evaluation** of expressions within some context;
it makes evaluation of the right depend on the evaluation of the left

https://www.quora.com/What-do-the-symbols-and-mean-in-haskell

Monad Definition

A **monad** is defined by

a **type constructor m**;
a function **return**;
an operator (**>>=**) "**bind**"

The function and operator are methods of the Monad type class and have types

```
return :: a -> m a

(>>=)  :: m a -> (a -> m b) -> m b
```

are required to obey three laws

https://en.wikibooks.org/wiki/Haskell/Understanding_monads

Monad Laws

every instance of the Monad type class must obey the following three laws:

```
m >>= return     =  m                          -- right unit
return x >>= f   =  f x                        -- left unit
(m >>= f) >>= g  =  m >>= (\x -> f x >>= g)     -- associativity
```

https://en.wikibooks.org/wiki/Haskell/Understanding_monads

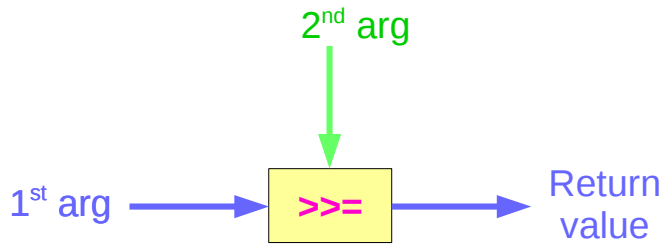
Monad Definition

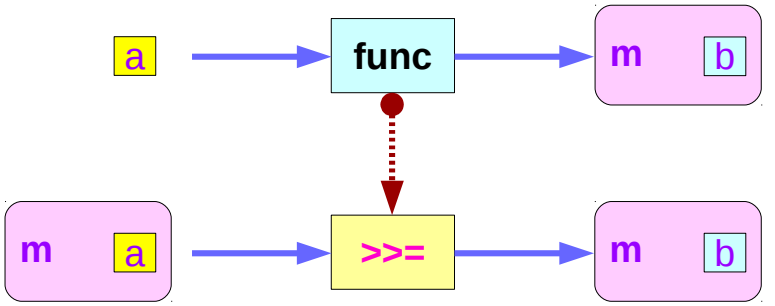
```
class Monad m where
    return :: a -> m a

    (>>=) :: m a -> (a -> m b) -> m b

    (>>) :: m a -> m b -> m b
    x >> y = x >>= \_ -> y

    fail :: String -> m a
    fail msg = error msg
```

https://en.wikibooks.org/wiki/Haskell/Understanding_monads

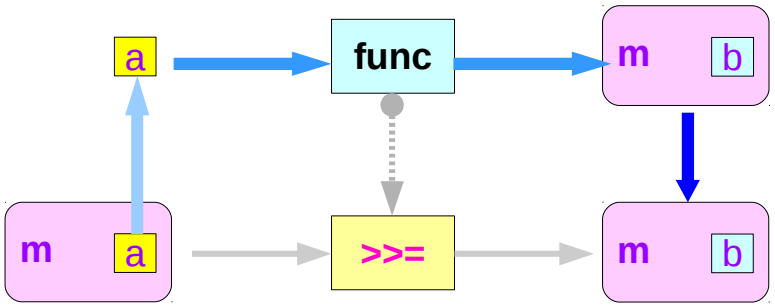
Monad – Bind Operation

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





1st arg	Monad	**m a**
2nd arg	Function	**(a -> m b)**
return	Monad	**m b**



Maybe Monad

the Maybe monad.

The **type constructor** is **m** = **Maybe**,

```
return :: a -> Maybe a

return x  = Just x
```

```
(>>=)  :: Maybe a -> (a -> Maybe b) -> Maybe b

m >>= g        = case m of
                   Nothing -> Nothing
                   Just x  -> g x
```

m :: **Maybe** a (Maybe monad)

g :: (a -> **Maybe** b) (function)

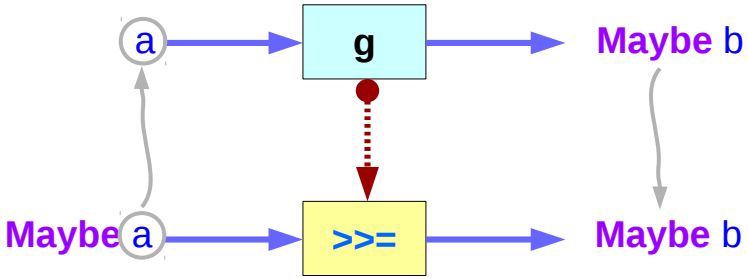
m >>= g (a function with 2 args)

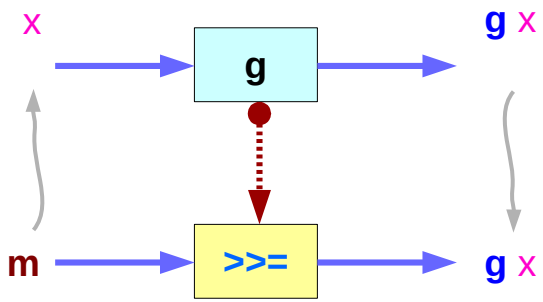
https://en.wikibooks.org/wiki/Haskell/Understanding_monads

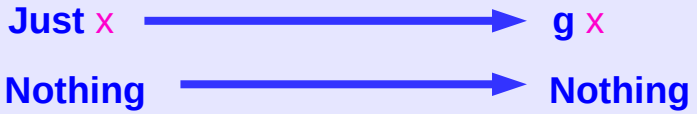
Monad Class Function >>= & >>

```
(>>=)  :: Maybe a -> (a -> Maybe b) -> Maybe b
```

```
m >>= g = case m of
            Nothing -> Nothing
            Just x  -> g x
```







https://en.wikibooks.org/wiki/Haskell/Understanding_monads

Monad Class Function >>= & >>

Maybe is the monad

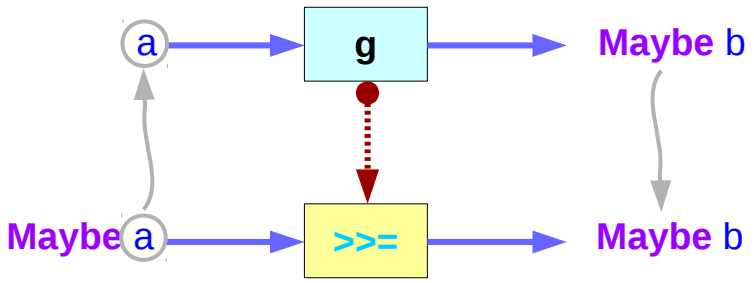
return brings a value into it
by wrapping it with **Just**

(>>=) takes
a value **m** :: **Maybe** a
a function **g** :: a -> **Maybe** b

if m is **Nothing**,
there is nothing to do and the result is **Nothing**.
Otherwise, in the **Just** x case,
the underlying value x is wrapped in **Just**
g is applied to x, to give a **Maybe** b result.

Note that this result *may* or *may not* be **Nothing**,
depending on what **g** does to x.

```
(>>=)  :: Maybe a -> (a -> Maybe b) -> Maybe b
m >>= g = case m of
            Nothing -> Nothing
            Just x  -> g x
```

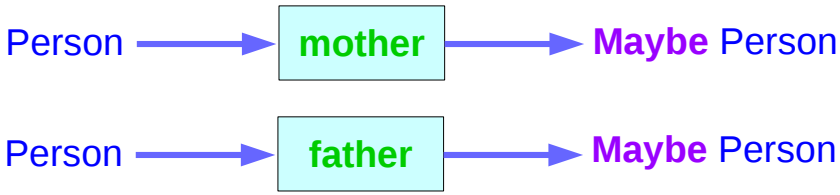


https://en.wikibooks.org/wiki/Haskell/Understanding_monads

Maybe Monad Examples

a family database that provides two functions:

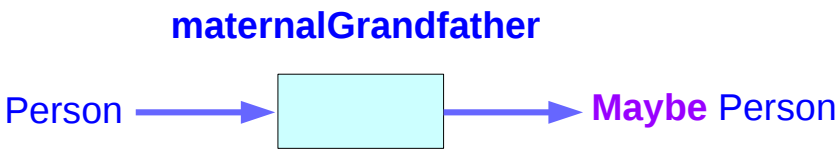
father :: Person -> **Maybe** Person
mother :: Person -> **Maybe** Person



Input the name of someone's father or mother.

If some relevant information is missing in the database
Maybe returns a **Nothing** value
to indicate that the lookup failed,
rather than crashing the program.

maternalGrandfather :: Person -> **Maybe** Person

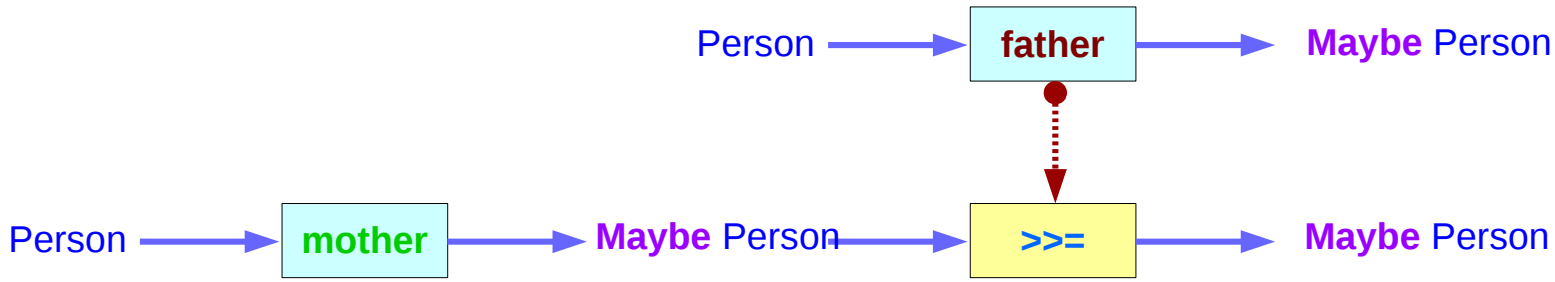


https://en.wikibooks.org/wiki/Haskell/Understanding_monads

Maybe Monad Examples

```
maternalGrandfather :: Person -> Maybe Person
maternalGrandfather p =
    case mother p of
        Nothing -> Nothing
        Just mom -> father mom
```

```
maternalGrandfather p = mother p >>= father
```



https://en.wikibooks.org/wiki/Haskell/Understanding_monads

Maybe Monad Examples

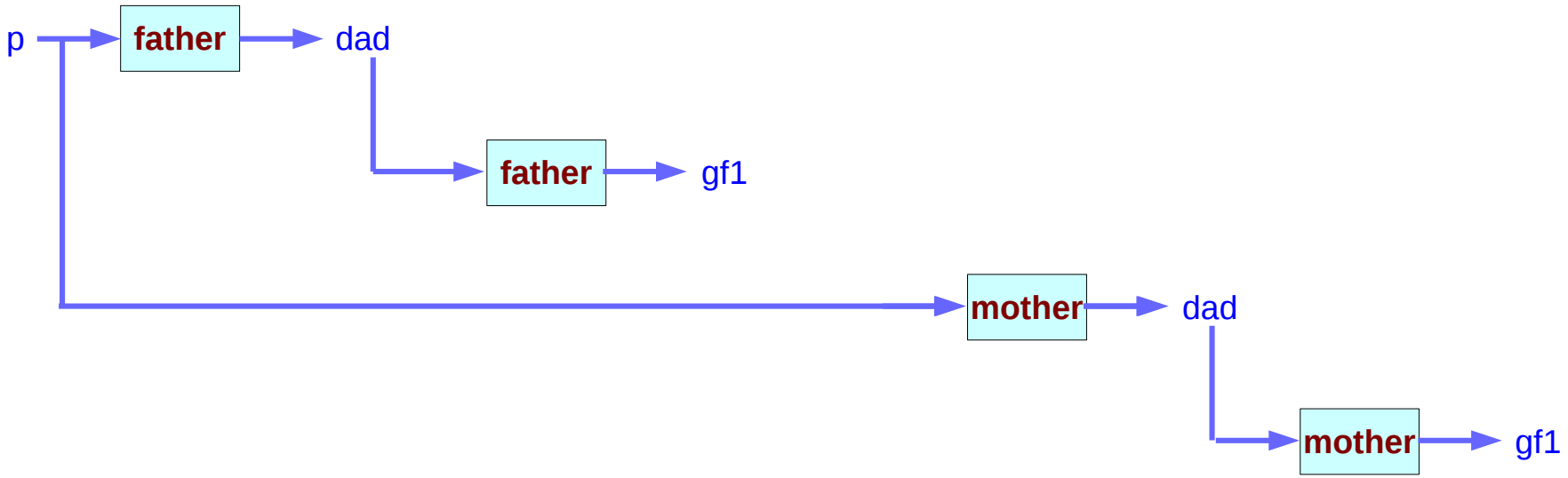
https://en.wikibooks.org/wiki/Haskell/Understanding_monads

```
bothGrandfathers :: Person -> Maybe (Person, Person)
bothGrandfathers p =
  case father p of
    Nothing -> Nothing
    Just dad ->
      case father dad of
        Nothing -> Nothing
        Just gf1 ->                    -- found first grandfather
          case mother p of
            Nothing -> Nothing
            Just mom ->
              case father mom of
                Nothing -> Nothing
                Just gf2 ->            -- found second grandfather
                  Just (gf1, gf2)
```

Maybe Monad Examples

https://en.wikibooks.org/wiki/Haskell/Understanding_monads

```
bothGrandfathers p =
  father p >>=
    (\dad -> father dad >>=
      (\gf1 -> mother p >>=   -- gf1 is only used in the final return
        (\mom -> father mom >>=
          (\gf2 -> return (gf1,gf2) ))))
```



Maybe Monad Examples

```
data Maybe a    = Just a
                | Nothing
```

a type definition: **Maybe** a

a parameter of a type variable a,

https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell

Maybe Monad Examples

```
data Maybe a     = Just a
                 | Nothing
```

two constructors: **Just** a and **Nothing**

a value of **Maybe** a type must be constructed via either **Just** or **Nothing**
there are no other (non-error) possibilities.

https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell

Maybe Monad Examples

```
data Maybe a    = Just a
                | Nothing
```

Nothing has no parameter type,
names a constant value
that is a member of type **Maybe** a for all types a.

Just constructor has a type parameter,
acts like a function from type a to **Maybe** a,
i.e. it has the type a -> **Maybe** a

https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell

Maybe Monad Examples

the (data) constructors of a type *build a value* of that type;

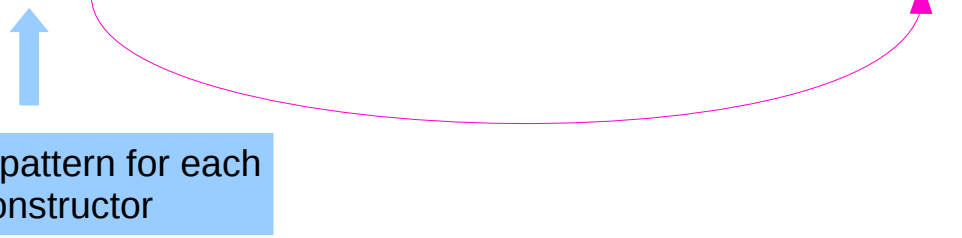
when using that *value*,
pattern matching can be applied

- Unlike functions, *constructors* can be used in *pattern binding expressions*
- **case analysis** of values that belong to types with more than one constructor.
- need to provide **a pattern** for each constructor

https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell

Maybe Monad Examples

```
case maybeVal of
  Nothing   -> "There is nothing!"
  Just val  -> "There is a value, and it is " ++ (show val)
```



a pattern for each constructor

https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell

Maybe

Maybe : Algebraic Data Type (ADT)

Widely used because it effectively extends a type Integer
into a new context in which it has an extra value (**Nothing**)
that represents *a lack of value*

check for that extra value before accessing the possible Integer

good for debugging

Many other languages have this sort of "**no-value**" value via NULL references.

The Haskel Maybe type handle this no-value more effectively.

https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell

Maybe as a functor

Functor type class:

- transforming one **type** to another
- transforming **operations** of one type to those of another

Maybe a has a useful instance of a **functor** type class

Functor provides **fmap** method

maps functions of the base type (such as Integer)

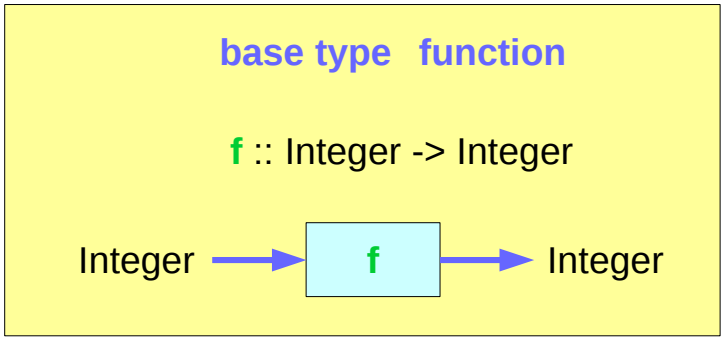
to *functions* of the lifted type (such as **Maybe** Integer).

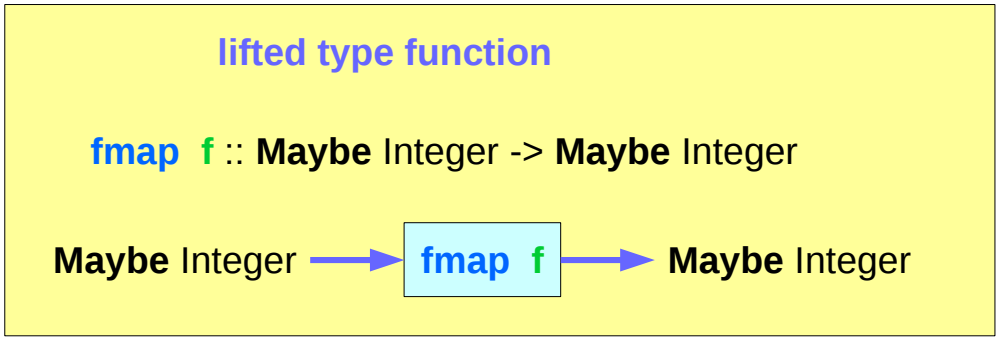
https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell

Maybe as a functor

A *function* **f** transformed with **fmap**
cab work on a Maybe value

```
case maybeVal of
  Nothing  -> Nothing      -- there is nothing, so just return Nothing
  Just val -> Just (f val) -- there is a value, so apply the function to it
```





https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell

Maybe as a functor

a **Maybe** Integer value: **m_x**

fmap f m_x

In fact, you could apply a whole chain of
lifted Integer -> **Integer** functions to **Maybe Integer** values
and only have to worry about explicitly checking for **Nothing** once when you're finished.

https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell

Maybe as a monad

the type signature **IO a** looks remarkably similar to **Maybe a**.

- IO doesn't expose its constructors
- only be "run" by the Haskell runtime system
- a Functor
- a Monad

a Monad is just a special kind of Functor with some extra features

value returning
Monads like **IO** *map* types to new types
that represent "computations that result in values"

lifting function
can *lift* **functions** into **Monad types**
via a very fmap-like function called **liftM**
that turns a regular function into a
"computation that results in the value obtained by evaluating the function."

https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell

Maybe as a monad

valueless return

Maybe is also a **Monad**
represents "computations that could *fail to return a value*"

no explicit check in each step

don’t have to check explicitly for errors after each step.

immediate abort

Because of the way the Monad instance is constructed,
a computation on Maybe values *stops as soon as* a **Nothing** is encountered,

https://stackoverflow.com/questions/18808258/what-does-the-just-syntax-mean-in-haskell

Monad – List Comprehension Examples

```
[x*2 | x<-[1..10], odd x]
```

```
do
  x <- [1..10]
  if odd x
     then [x*2]
     else []
```

```
[1..10] >>= (\x -> if odd x then [x*2] else [])
```

https://stackoverflow.com/questions/44965/what-is-a-monad

Monad – I/O Examples

```
do
   putStrLn "What is your name?"
   name <- getLine
   putStrLn ("Welcome, " ++ name ++ "!")
```

https://stackoverflow.com/questions/44965/what-is-a-monad

Monad – A Parser Example

```
parseExpr = parseString <|> parseNumber

parseString = do
        char ""
        x <- many (noneOf "\"")
        char ""
        return (StringValue x)

parseNumber = do
    num <- many1 digit
    return (NumberValue (read num))
```

https://stackoverflow.com/questions/44965/what-is-a-monad

Monad – Asynchronous Examples

```
let AsyncHttp(url:string) =
    async {  let req = WebRequest.Create(url)
             let! rsp = req.GetResponseAsync()
             use stream = rsp.GetResponseStream()
             use reader = new System.IO.StreamReader(stream)
             return reader.ReadToEnd() }
```

https://stackoverflow.com/questions/44965/what-is-a-monad

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf