Monoid (4A)

Young Won Lim
5/8/18

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

This document was produced by using LibreOffice.

Young Won Lim
5/8/18

Based on

http://learnyouahaskell.com/making-our-own-types-and-typeclasses#the-functor-typeclass

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

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

Binary Operation

associativity

```
GHCi> (5 + 6) + 10
21
GHCi> 5 + (6 + 10)
21
```

```
GHCi> (5 * 6) * 10
300
GHCi> 5 * (6 * 10)
300
```

```
GHCi> ([1,2,3] ++ [4,5,6]) ++ [7,8,9]
[1,2,3,4,5,6,7,8,9]
GHCi> [1,2,3] ++ ([4,5,6] ++ [7,8,9])
[1,2,3,4,5,6,7,8,9]
```

an identity element

```
GHCi> 255 + 0
255
GHCi> 0 + 255
255
```

```
GHCi> 255 * 1
255
GHCi> 1 * 255
255
```

```
GHCi> [1,2,3] ++ []
[1,2,3]
GHCi> [] ++ [1,2,3]
[1,2,3]
```

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

Monoid Type Definition

```
class Monoid m where
  mempty :: m
  mappend :: m -> m -> m
  mconcat :: [m] -> m
  mconcat = foldr mappend mempty
```

https://wiki.haskell.org/Currying
http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Currying

```
mempty `mappend` x = x
x `mappend` mempty = x
(x `mappend` y) `mappend` z = x `mappend` (y `mappend` z)
```

```
class Monoid m where
    mempty :: m
    mappend :: m -> m -> m
    mconcat :: [m] -> m
    mconcat = foldr mappend mempty
```

https://wiki.haskell.org/Currying
http://learnyouahaskell.com/functors-applicative-functors-and-monoids

List Monoid Examples

```
ghci> [1,2,3] `mappend` [4,5,6]
[1,2,3,4,5,6]
ghci> ("one" `mappend` "two") `mappend` "tree"
"onetwotree"
ghci> "one" `mappend` ("two" `mappend` "tree")
"onetwotree"
ghci> "one" `mappend` "two" `mappend` "tree"
"onetwotree"
ghci> "pang" `mappend` mempty
"pang"
ghci> mconcat [[1,2],[3,6],[9]]
[1,2,3,6,9]
ghci> mempty :: [a]
[]
```

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Product and Sum Monoid Examples

```
ghci> getProduct $ Product 3 `mappend` Product 9
27
ghci> getProduct $ Product 3 `mappend` mempty
3
ghci> getProduct $ Product 3 `mappend` Product 4 `mappend` Product 2
24
ghci> getProduct . mconcat . map Product $ [3,4,2]
24
```

```
ghci> getSum $ Sum 2 `mappend` Sum 9
11
ghci> getSum $ mempty `mappend` Sum 3
3
ghci> getSum . mconcat . map Sum $ [1,2,3]
6
```

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Any and All Monoid Examples

```
ghci> getAny $ Any True `mappend` Any False
True
ghci> getAny $ mempty `mappend` Any True
True
ghci> getAny . mconcat . map Any $ [False, False, False, True]
True
ghci> getAny $ mempty `mappend` mempty
False
```

```
ghci> getAll $ mempty `mappend` All True
True
ghci> getAll $ mempty `mappend` All False
False
ghci> getAll . mconcat . map All $ [True, True, True]
True
ghci> getAll . mconcat . map All $ [True, True, False]
False
```

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Ordering Monoid Examples

```
ghci> LT `mappend` GT
LT
ghci> GT `mappend` LT
GT
ghci> mempty `mappend` LT
LT
ghci> mempty `mappend` GT
GT
```

```
ghci> lengthCompare "zen" "ants"
LT
ghci> lengthCompare "zen" "ant"
GT

ghci> lengthCompare "zen" "anna"
LT
ghci> lengthCompare "zen" "ana"
LT
ghci> lengthCompare "zen" "ann"
GT
```

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Maybe Monoid Examples

```
ghci> Nothing `mappend` Just "andy"
Just "andy"
ghci> Just LT `mappend` Nothing
Just LT
ghci> Just (Sum 3) `mappend` Just (Sum 4)
Just (Sum {getSum = 7})

ghci> getFirst $ First (Just 'a') `mappend` First (Just 'b')
Just 'a'
ghci> getFirst $ First Nothing `mappend` First (Just 'b')
Just 'b'
ghci> getFirst $ First (Just 'a') `mappend` First Nothing
Just 'a'

ghci> getFirst . mconcat . map First $ [Nothing, Just 9, Just 10]
Just 9

ghci> getLast . mconcat . map Last $ [Nothing, Just 9, Just 10]
Just 10
ghci> getLast $ Last (Just "one") `mappend` Last (Just "two")
Just "two"
```

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Folding and Monoids Examples (1)

```
ghci> :t foldr
foldr :: (a -> b -> b) -> b -> [a] -> b
ghci> :t F.foldr
F.foldr :: (F.Foldable t) => (a -> b -> b) -> b -> t a -> b

ghci> foldr (*) 1 [1,2,3]
6
ghci> F.foldr (*) 1 [1,2,3]
6

ghci> F.foldl (+) 2 (Just 9)
11
ghci> F.foldr (||) False (Just True)
True

ghci> F.foldl (+) 0 testTree
42
ghci> F.foldl (*) 1 testTree
64800
```

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

Folding and Monoids Examples (2)

```
ghci> getAny $ F.foldMap (\x -> Any $ x == 3) testTree
True

ghci> getAny $ F.foldMap (\x -> Any $ x > 15) testTree
False

ghci> F.foldMap (\x -> [x]) testTree
[1,3,6,5,8,9,10]
```

http://learnyouahaskell.com/functors-applicative-functors-and-monoids

<> : infix synonym for **mappend**

```
class Monoid m where
  mempty :: m
  mappend :: m -> m -> m
  mconcat :: [m] -> m
  -- defining mconcat is optional, since it has the following default:
  mconcat = foldr mappend mempty

-- this infix synonym for mappend is found in Data.Monoid
x <> y = mappend x y
infixr 6 <>
```

together with the following laws:

```
-- Identity laws
x <> mempty = x
mempty <> x = x

-- Associativity
(x <> y) <> z = x <> (y <> z)
```

https://wiki.haskell.org/Monoid

References

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