Gathered notes from:

- Haskell Programming from First Principles [1]

1 Monoid

todo

2 Functor

todo

3 Applicative

todo

4 Monad

todo

5 Arrow operator as Functor and Applicative

5.1 Functor

`((->) r) = (r ->)` as a functor
`(r ->) *` expects a type as argument
instances of `(->) r *` as a type class
examples of other functors: `[] *`, `Maybe *`
functor as a type constructor

fmap:
```
<$>:: (a -> b) -> F a -> F b
```
let `F = (->) r`
```
<$>:: (a -> b) -> ((->) r) a -> ((->) r) b
<$>:: (a -> b) -> (r -> a) -> (r -> b)
<$>:: (a -> b) -> (r -> a) -> r -> b
```

composition operator:
```
(.) :: (b -> c) -> (a -> b) -> a -> c
```

therefore,
`<$> = (.)` where `F = (->) r` for functor

5.1.1 example

```
(+) <$> (*2)
(+) . (*2)
\x -> (+) ((*2) x)
\x -> (+) (x*2)
\x -> x*2 :: a -> a
(\x -> (+) (x*2)) :: a -> (a -> a)
(\x -> (+) (x*2)) :: a -> a -> a
(\x -> ((x*2)+) :: a -> a -> a
(\x -> (\y -> (x*2) + y) :: a -> a -> a
```

5.2 Applicative

apply:
```
<*>:: F (a -> b) -> F a -> F b
```
let `F = (->) r = r ->`, then
```
<*>:: ((->) r) (a->b) -> ((->) r) a -> ((->) r) b
<*>:: (r -> a -> b) -> (r -> a) -> (r -> b)
```

```
pure :: a -> F a
pure x = ((->) r) x = r -> x :: F a
```

5.2.1 example

```
(+) <$> (*2) <*> (+10)
(+) . (*2) <*> (+10)
(\x -> (+) (x*2)) <*> (\x -> x + 10)
\x -> (+) (x*2) (x+10)
```

types:
`\x -> :: (->) r`
`(+) (x*2) :: a -> b` where `x` is fixed
`\x -> (+) (x*2) :: ((->) r) a -> b`

```
\x -> x + 10 :: r -> a  = ((->) r) a
```

`(+) (x*2) (x+10) :: b` where `x` is fixed
`\x -> (+) (x*2) (x+10) :: ((->) r) b`

```
<*> :: (((->) r) a -> b) -> (((->) r) a) -> (((->) r) b)
```

thus types are as expected for applicative

References

[1] Allen & Moronuki. Haskell programming from first principles, 2016.