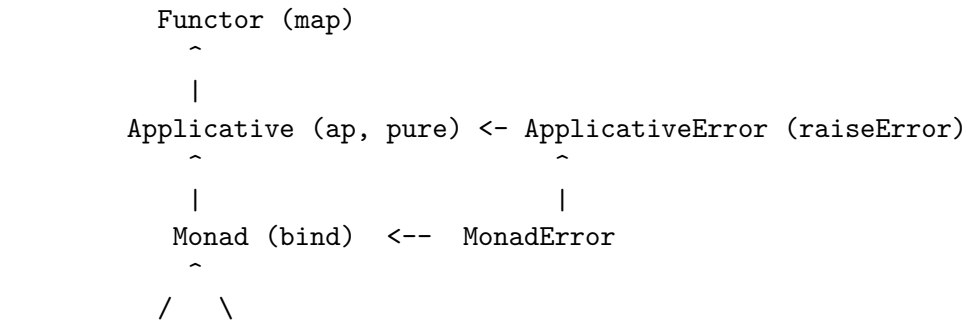
50.054 Functor, Applicative and Monad

ISTD, SUTD

Recap Monad

- List Monad
- `Option` Monad and `EitherErr` Monad
- `Reader` Monad (with `ReaderReader`)
- `State` Monad (with `StateMonad`)

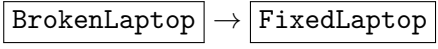
Recap Functor, Applicative and Monad



What is Functor, Applicative and Monad?

What is Functor?

- the CS Dude



- your secret contractor

$\texttt{BrokenLaptop} \to \texttt{FixedLaptop}$

What is Monad?

- the CS Dude

`BrokenLaptop` → `FixedLaptop`

- your reliable contractor

`BrokenLaptop` → `FixedLaptop`

What is Applicative?

- the CS Dude

`BrokenLaptop` $\rightarrow$ `FixedLaptop`

- your  contractor

```
BrokenLaptop -> FixedLaptop
```

Recap Reader Monad

```
case class Reader[R, A] (run: R=>A) {
    // we need flatMap and map for for-comprehension
    def flatMap[B](f:A =>Reader[R,B]):Reader[R,B] = this match {
        case Reader(ra) => Reader (
            r => f(ra(r)) match {
                case Reader(rb) => rb(r)
            }
        )
    }
    def map[B](f:A=>B):Reader[R, B] = this match {
        case Reader(ra) => Reader (
            r => f(ra(r))
        )
    }
}
```

Recap Reader Monad

```
type ReaderM = [R] =>> [A] =>> Reader[R, A]

trait ReaderMonad[R] extends Monad[ReaderM[R]] {
    override def pure[A](v:A):Reader[R, A] = Reader (r => v)
    override def bind[A,B](fa:Reader[R, A])(f:A=>Reader[R,B]):Reader[R,B] =
        fa.flatMap(f)
    def ask:Reader[R,R] = Reader( r => r)
    def local[A](f:R=>R)(r:Reader[R,A]):Reader[R,A] = r match {
        case Reader(ra) => Reader( r => {
            val localR = f(r)
            ra(localR)
        })
    }
}
```

Recap Reader Monad

- Let's say our monad is `ReaderMonad[BrokenLaptop]`
- Hence the container type is `[A] =>> Reader[BrokenLaptop, A]`

```
def callYourContractor (a : BrokenLaptop) : Cost
    = ...

val quote : Reader[BrokenLaptop, Cost] =
    Reader(callYourContractor)

def sendToContractor(a : BrokenLaptop)
    : FixedLaptop = ...

def getItFixed(c:Cost)
    : Reader[BrokenLaptop, FixedLaptop] =
    Reader(sendToContractor)

def yourTask(using i: ReaderMonad[BrokenLaptop])
    : Reader[BrokenLaptop, FixedLaptop] = for {
    cost   <- quote
    laptop <- getItFixed(cost)
} yield laptop
```

Recap State Monad

```
case class State[S,A]( run:S=>(S,A)) {
    def flatMap[B](f: A => State[S,B]):State[S,B] = this match {
        case State(ssa) => State(
            s=> ssa(s) match {
                case (s1,a) => f(a) match {
                    case State(ssb) => ssb(s1)
                }
            }
        )
    }
    def map[B](f:A => B):State[S,B] = this match {
        case State(ssa) => State(
            s=> ssa(s) match {
                case (s1, a) => (s1, f(a))
            }
        )
    }
}
```

Recap State Monad

```
case class State[S,A]( run:S=>(S,A)) {
    def flatMap[B](f: A => State[S,B]):State[S,B] = this match {
        case State(ssa) => State(
            s=> ssa(s) match {
                case (s1,a) => f(a) match {
                    case State(ssb) => ssb(s1)
                }
            }
        )
    }
    def map[B](f:A => B):State[S,B] = this match {
        case State(ssa) => State(
            s=> ssa(s) match {
                case (s1, a) => (s1, f(a))
            }
        )
    }
}
```

Recap State Monad

- Let's say our monad is `StateMonad[Money]`
- Hence the container type is `[A] =>> State[Money, A]`

```
def getItForFree(a : BrokenLaptop)(using i:StateMonad[Money])
    : State[Money, BrokenLaptop] = i.pure(a)

def fixKeyboard (a : BrokenLaptop)(using i:StateMonad[Money])
    : State[Money, LaptopWithBrokenScreen]
    = ... // with some money deducted via i.get and i.set

val fixScreen (a : LaptopWithBrokenScreen)(using i:StateMonad[Money])
    : State[Money, FixedLaptop]
    = ... // with some money deducted via i.get and i.set

def yourTask(laptop:BrokenLaptop)(using i:StateMonad[Money])
    : State[Money, FixedLaptop] = for {
    laptop_with_state1  <- getItForFree(laptop)
    laptop_with_state2  <- fixKeyboard(laptop_with_state1)
    laptop_with_state3  <- fixScreen(laptop_with_state2)
} yield laptop_with_state3
```