HASKELL MONADS

Curt Clifton Rose-Hulman Institute of Technology

MONADS

- Ooh, scary!
- Not really, just an extremely useful example of generalization
- Goal: recognize monads as a general solution to lots of problems



Lon Chaney, Jr. as The Wolf Man

CAN WE BE JUST A LITTLE BIT IMPURE?

Haskell is a pure, functional language

Consider: random :: Int

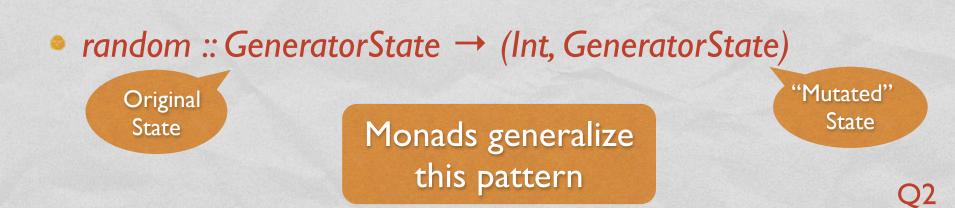
RANDOM NUMBER

int getRandomNumber() { return 4; // chosen by fair dice roll. // guaranteed to be random. }

RFC 1149.5 specifies 4 as the standard IEEE-vetted random number.

CAN WE BE JUST A LITTLE BIT IMPURE?

- Haskell is a pure, functional language
- Consider: random :: Int
- Solution: Pass along an object to be "mutated"



THREADING STATE

- Haskell's lazy, so...
 - Infinite lists
 - randomList :: GeneratorState -> [Int] randomList state = x : randomList nextState where (x, nextState) = random state
- Benefits:

Söylemez calls this "threading state"

- Can "go back in time" to earlier random numbers
- Can pass the same sequence to multiple functions

WHAT ABOUT IO? Thought experiment

- getChar :: Char
- getChar :: Universe -> (Char, Universe)

 twoChars :: Universe -> (Char, Char, Universe) twoChars world0 = (c1, c2, world2) where (c1, world1) = getChar world0 (c2, world2) = getChar world1

strangeDays :: Universe -> (Result, Result)
strangeDays world = (c1, c2)
where (r1, _) = killCat world
(r2, _) = freeCat world

Threading the state of the universe leads to paradoxes

MOTIVATION

• Can we generalize this idea of passing state around without doing it directly?





EXAMPLES

isqrt :: Integer -> Maybe Integer isqrt x = isqrt' x (0,0)where isqrt' x (s,r) | s > x = Nothing | s == x = Just r| otherwise = isqrt' x <math>(s + 2*r + 1, r+1)

i4throot :: Integer -> Maybe Integer i4throot x = case isqrt x of Nothing -> Nothing Just y -> isqrt y

> Maybe computation made of Maybe computations

EXAMPLES

isqrtL :: Integer -> [Integer] isqrtL x = isqrt' x (0,0) where isqrt' x (s,r) | s > x = [] | s = x = [r, -r]otherwise = isqrt' x (s + 2*r + 1, r+1)

i4throotL :: Integer -> [Integer] i4throotL x = case isqrtL x of [] -> [] [y, _] -> isqrtL y

> List computation made of List computations

GENERAL IDEA

Start and the Start ton

- A computation with a certain type of result
 - e.g., Integer
- A certain type of structure in its result
 - e.g., Nothing, [], [2, -2]
- Need to pass the result of one of these computations to another

Monads let us build up these computations as static entities without necessarily running them

MONAD TYPECLASS

Sector States and the sector of the

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

return takes a value of the inner type and wraps it in a computation

binding operator takes a computation and feeds its value to a function that makes a another computation

MAYBE AS A MONAD

Contractorious a contract of Prane torne

 instance Monad Maybe where return x = Just x return takes a value of the inner type and wraps it in a computation

Nothing >>= f = Nothing Just x >>= f = f x

binding operator takes a computation and feeds its value to a function that makes a another computation

LIST AS A MONAD

State Black and Black the

instance Monad [] where return x = [x] return takes a value of the inner type and wraps it in a computation

xs >>= f = concat (map f xs)

binding operator takes a computation and feeds its value to a function that makes a another computation

[10,20,40] >>= \x -> [x+1, x+2]

NEXTTIME

Monads for combining computations that use state