Week 2
Other kinds of Monads
Monads

Maybe

IO is not the only monad.

Here is a datatype that might be familiar to you:

```haskell
data Maybe a = Just a | Nothing
```
A typical use of the Maybe type would involve some partial function. Say, a dictionary lookup which could fail. Here’s a function that tries to look up two separate keys and return a pair of the results (or return Nothing if either lookup fails)

```haskell
findEntry :: Tree a b -> a -> Maybe b
findentry = …

f tree = case (findEntry "foo" tree) of
  Nothing -> Nothing
  Just x  -> case (findEntry "bar" tree) of
              Nothing -> Nothing
              Just y  -> Just (x,y)
```
Monads

Maybe

Interesting fact: “Maybe” is a monad!

That means there is a definition of “$$\gg=\$$” for it, and we can rewrite “f” as:

```haskell
f tree = do
    x <- findEntry "foo" tree
    y <- findEntry "bar" tree
    return (x,y)
```
Monads

Maybe is a Monad!

When I say Maybe is a Monad, I mean that it has

- return :: Maybe a
- bind :: Maybe a \to (a \to Maybe b) \to Maybe b

```
return x = Just x

Nothing >>= k = Nothing
(Just x) >>= k = k x
```
Monads

Maybe is a Monad!

• If we de-sugar “f” so that we can see the “>>=” operations it makes it easier to see how this works:

\[
\begin{align*}
\text{findEntry “foo” tree} & \gg= (\lambda x \rightarrow \\
\text{findEntry “bar” tree} & \gg= (\lambda y \rightarrow \\
& \text{return } (x, y))
\end{align*}
\]

• Say the first “findEntry” returns nothing. Then we have:

\[
\begin{align*}
\text{Nothing} & \gg= (\lambda x \rightarrow \\
\text{findEntry “bar” tree} & \gg= (\lambda y \rightarrow \\
& \text{return } (x, y))
\end{align*}
\]

• By the first equation of (gg=), that whole expression immediately becomes:

\[
\text{Nothing}
\]
Monads

How can there be two Monads?

• How is it that we have both IO as a monad and Maybe as a monad?
• The definition for “return” cannot both be something talking about World and something talking about Just?
• The solution lies in Haskell’s Class Mechanism which lets us do a kind of structured overloading.
Thank you

glenn.Strong@scss.tcd.ie
https://scss.tcd.ie/Glenn.Strong/