Curtis Millar
CSE, UNSW (and Data61)
15 July 2020

Exercise 4

- Capitalise a

Exercise 4

- Capitalise a
- Sum all the nu

Exercise 4

- Capitalise a
- Sum all the nu
- Implement a guessing game AI.

State & IO

Week 5 cover

Functors, Applicatives, Monads

- Consider *higher-kinded* types of kind `* -> *` that *contain* or *produce* their argument `t`

Functors, Applicatives, Monads

- Consider *higher-kinded* types of kind `* -> *` that *contain* or *produce* their argument `t`
- *Functor* le
to different c

Functors, Applicatives, Monads

- Consider *higher-kinded* types of kind `* -> *` that *contain* or *produce* their argument `t`
- *Functor* le

to different c
- *Applicative* lets us apply a *n*-ary function in the c e.
- *Monad* lets us *sequentially compose* functio

higher-kinded type.

Functors


```
class Functor f where
  fmap :: (a -> b) -> f a -
```

The functor type cl

Functors


```
class Functor f where
  fmap :: (a -> b) -> f a -
```

The functor type c

Functor Laws

1. `fmap id == id`
2. `fmap f . fmap g == fmap (f . g)`

Applicatives


```
class Functor f => Applicative f where
  pure :: a -> f a
  (<*>) :: f (a -> b) -
```

The functor type cl

Applicatives


```
class Functor f => Applicative f where
  pure :: a -> f a
  (<*>) :: f (a -> b) -
```

The functor type cl

Applicative La

1. `pure id <*> v = v` (Identity)
2. `pure f <*> pure x = pure (f x)` (H
3. `u <*> pure y = pure ($ y) <*> u` (Int
4. `pure (.) <*> u <*> v <*> w = u <*> (v <*> w)` (Composition)

Alternative Applicative

It is possible to express Applicative equivalently as:

```
class Functor f => App f where
  pure :: a -> f a
  tuple :: f a -> f b -> f (a
```


Example (Alter

Alternative Applicative

It is possible to express Applicative equivalently as:

```
class Functor f => App f where
  pure :: a -> f a
  tuple :: f a -> f b -> f (a
```


Example (Alter

1. Using `tuple`, `fmap` and `pure`, let's implement
2. And, using `<*>`, `fmap` and `pure`, let's impleme

done in Haskell.

Alternative Applicative

It is possible to express Applicative equivalently as:

```
class Functor f => App f where
  pure :: a -> f a
  tuple :: f a -> f b -> f (a
```


Example (Alter

1. Using `tuple`, `fmap` and `pure`, let's implement
2. And, using `<*>`, `fmap` and `pure`, let's impleme

done in Haskell.

Proof exercise: Prove that `tuple` obeys the applicative laws.

Monads

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

We can define a com

Monads


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

We can define a com

```
(<=<) :: (b -> m c) -> (a -
(f <=< g) x = g x >>= f
```


The monad type class must obey three additional laws:

Monad Laws

1. `f <=< (g <=< x) == (f <=< g) <=< x`
2. `pure <=< f == f` (left identity)
3. `f <=< pure == f` (right identity)

Alternative Monad

It is possible to express Monad equivalently as:

```
class Applica
  join :: m (m a) -> m a
```


Example (Alter

Alternative Monad

It is possible to express Monad equivalently as:

```
class Applica
  join :: m (m a) -> m a
```


Example (Alter

1. Using `join` and `fmap`, let's implement >>=.
2. And, using >>= let's implement `join`.

done in Haskell.

Tree Example


```
data Tree a
   = Leaf
   | Node a (Tree a
    derivin
```


Example (Tree Example)

Show that `Tree` is an `Applicative` instance.
done in Haskell:

Tree Example


```
data Tree a
   = Leaf
   | Node a (Tree a
    derivin
```


Example (Tree Example)

Show that `Tree` is an `Applicative` instance.
done in Haskell!

Note that `Tree` is not a `Monad` instance.

Formulas Example


```
data Formula v = Var v
               | And (Formula v) (Formula v)
```



```
               deriving (Eq,Show)
```


Example (Formulas Example)

Show that `Formula` is a `Monad` instance.

done in Haskell.

Homework

Homework

1. Week 5's qui

Homework

1. Week 5's qui
2. The fifth pro

Homework

1. Week 5's qui
2. The fifth pro
3. This week's quiz is also up, it's due Friday week (in 9 days).

Consultations

- Consultations will be made on request. Ask on piazza or email `cs3141@c`

Consultations

- Consultations will be made on request. Ask on piazza or email cs3141@c
- If there is a con number for

Consultations

- Consultations will be made on request. Ask on piazza or email cs3141@c
- If there is a con
number for

- Will be in the Thursday lecture slot, 9am to 11am on Blackb

Consultations

- Consultations will be made on request. Ask on piazza or email `cs3141@c`
- If there is a con

number for
- Will be in the Thursday lecture slot, 9am to 11am on Blackb
- Make sure to join the queue on Hopper. Be ready to share you

(`ghci` or `stack repl`) and editor set up.