4798138

Hyeyeon Lee

A3 written

Either

The Functor and Applicative for the `Either` data-type is as follows:

```
instance Functor (Either a) where
    fmap f (Right x) = Right (f x)
    fmap f (Left x)  = Left x

instance Applicative (Either e) where
    pure            =  Right
    Left  e  <*>  _  =  Left e
    Right f  <*>  r  =  fmap f r
```

Question 1. [4 MARKS]

Show `Either` satisfies the *second* functor law:

```
fmap (g . h) = fmap g . fmap h
```

Application of composition

(f.g) x = f.g $ x = f $ g x = f $ g $ x

For any x :: Either

P(x) ⟷ (g.h) <$> x = (g <$>).(h <$>) x

Suppose x = Right a

(g.h) <$> (Right a)

= Right (g.h $ a) apply (g.h)<$>

= Right (g $ h $ a) composition

= (g <$>) Right (h $ a) unapply g <$>

= (g <$>) $ (h <$>) Right a unapply h<$>

= ((g <$>) (h <$>)) (Right a) unapply composition

Therefore, P(Right a)

Suppose x = Left a

(g.h) <$> (Left a)

= Left a apply (g.h) <$>

= (g <$>) Left a unapply g <$>

= (g<$>) $ (h <$>) Left a unapply h <$>

= ((g <$>).(h <$>)) Left a unapply composition

Therefore, P(Left a)

Either

The Functor and Applicative for the `Either` data-type is as follows:

```
instance Functor (Either a) where
    fmap f (Right x) = Right (f x)
    fmap f (Left x)  = Left x

instance Applicative (Either e) where
    pure             =  Right
    Left  e  <*>  _  =  Left e
    Right f  <*>  r  =  fmap f r
```

Question 2. [6 MARKS]

Show `Either` satisfies the *third* applicative law:

```
x <*> pure y = pure (\g -> g y) <*> x
```

x <*> pure y = pure ($y) <*> x

Assuming x = Right a

pure ($y) <*> Right a

= ($y) <$> Right a   apply <*>

= Right (($y) a)   apply <$>

= Right (a $ y)   apply ($y)

= a <$> Right y   unapply <$>

= Right a <*> Right y   unapply <*>

= x <*> Right y   by assumption

= x <*> pure y   unapply pure

ZipWith

Recall the alternate definition for a list applicative given in tutorial 8.

```
1 instance Functor [] where
2     fmap _ []     = []
3     fmap g (x:xs) = g x : (fmap g xs)
4
5 instance Applicative [] where
6     pure f = repeat f
7     [] <*> _ = []
8     _ <*> [] = []
9     (f:fs) <*> (x:xs) = (f x) : (fs <*> xs)
```

When writing your proofs *use the line numbers given above* when justifying your steps.

Question 3. [7 MARKS]

Show your Applicative satisfies the *second* applicative law:

```
pure (g x) = pure g <*> pure x
```

Base case   pure (g [])   P(x) : Pure (g x) = pure g <*> pure x

= []   line 2   apply g

= (pure g) <*> []   unapply (pure g) <*>

= (pure g) <*> (pure [])   unapply pure

Therefore P([])

Induction   Suppose P(xs) and prove P(x:xs)   HI   P(xs) : pure (g xs) = pure g <*> pure xs

Pure g <*> Pure (x:xs)   repeat (g xs) = repeat g <*> repeat xs

= repeat g <*> repeat (x:xs)   apply pure   Definition of repeat : repeat x = x : (repeat x)

= repeat g <*> (x : repeat xs)   definition of repeat

= repeat (g $ x) : repeat g <*> repeat xs   apply <*>

= repeat (g $ x) : repeat (g $ xs)   apply <*> + P(xs) HI

= repeat (g (x:xs))   apply g $

Therefore P(x:xs)

ZipWith

Recall the alternate definition for a list applicative given in tutorial 8.

```
1 instance Functor [] where
2     fmap _ []     = []
3     fmap g (x:xs) = g x : (fmap g xs)
4
5 instance Applicative [] where
6     pure f = repeat f
7     [] <*> _ = []
8     _ <*> [] = []
9     (f:fs) <*> (x:xs) = (f x) : (fs <*> xs)
```

When writing your proofs *use the line numbers given above* when justifying your steps.

Question 4. [8 MARKS]

Show your Applicative satisfies the *forth* applicative law:

```
x <*> (y <*> z) = (pure (.) <*> x <*> y) <*> z
```

Base case

x= [] y = bs z = cs

[] <*> (bs <*> cs)

= [] line 7 apply <*>

= [] <*> (bs) line 7 unapply <*> bs

= (pure (.) <*> [] <*> bs) line 8 unapply pure (.) <*>

= (pure (.) <*> [] <*> bs) <*> cs line 7 unapply <*> cs

Loss of Generality, Same progress when y = [] or z = []

x <*> (y <*> z) = (pure (.) <*> x <*> y) <*> z is true for Base case when one of the input is empty.

Induction	x= (a:as) y= (b:bs) z= (c:cs)	
	x <*> (y <*> z) = (pure (.) <*> x <*> y) <*> z	
	Presume as <*> (bs <*> z) = (pure (.) <*> as <*> bs) <*> cs	
	(pure (.) <*> (a:as) <*> (b:bs) <*> (c:cs)	
	((.) a : repeat (.) <*> as) <*> (b:bs) <*> (c:cs)	apply <*>
	(((.) a b) : (repeat (.) <*> as) <*> bs)) <*> (c:cs)	apply <*>
	(a.b) c : (((repeat (.) <*> as) <*> bs) <*> cs)	apply <*>
	(a.b) c : as <*> bs <*> cs	by HI
	(a $ b $ c) : as <*> bs <*> cs	composition
	(a:as) <*> ((b$c) : (bs <*> cs))	line 9 unapply <*>
	(a:as) <*> ((b:bs) <*> (c:cs))	line 9 unapply <*>
	Therefore , (a:as) <*> ((b:bs) <*> (c:cs))	
	= (pure (.) <*> (a:as)) <*> (b:bs) <*> (c:cs)	
	= x <*> (y <*> z)	