Name:

Yinxuan Feng

BU ID (no dashes):

U17037273

CS 320—Homework Four—Due 2/25 at 11:59PM

Write your answers to the problems in the space indicated. Scan your solution and submit to Gradescope as a PDF file. You will receive an email about the Gradescope account. You may do this from your phone using free scanning apps, or with a desktop scanner. Do NOT edit this file and move things around, the format must remain the same.

Problem One (Eq Laws: A Good Instance)

In order for type classes to work properly in the Haskell ecosystem, they have to follow certain algebraic properties, which are called "laws" in the Haskell community. The first three Eq Laws are the following (these are just the axioms of an equivalence relation for math nerds—you know who you are!):

Reflexivity: ∀x. x == x

Symmetry: ∀x, y. x == y ↔ y == x

Transitivity: ∀x, y, z. x == y ∧ y == z → x == z

In this problem we will look at an example of a potential instance of Eq which satisfies these laws (a *good* instance); in the next we'll look at one which doesn't (a *bad* instance).

Note: In general, when doing these kinds of proofs, you proceed by structural induction (remember this from CS 131?). Here the "induction" is not really necessary, since Pairs are not a recursive type, but you'll have to use the fact that the types a and b DO satisfy the laws in such a proof (if necessary).

Consider the following instance of the Functor Eq:

```
data Pair a b = P a b

instance (Eq a, Eq b) => Eq (Pair a b) where
  -- (==) :: Pair a b -> Pair a b -> Bool
  (P x y) == (P x' y') = x == x' && y == y'
```

(a) Prove that this satisfies Reflexivity.

~~φ~~ x = Pair a b

a == a   b == b

∴ x == x

(b) Prove that this satisfies Symmetry.

x = Pair a_1 b_1 y = Pair a_2 b_2

if $a_1 == a_2$ and $b_1 == b_2$ then $a_2 == a_1$ and $b_2 == b_1$,

∴ if x == y then y == x, if y == x then x == y

(c) Prove that this satisfies Transitivity.

x = Pair a_1 b_1 y = Pair a_2 b_2 z = Pair a_3 b_3

if x == y ∧ y == z then $a_1 == a_2 \wedge b_1 == b_2$ ~~∧~~ $a_2 == a_3 \wedge b_2 == b_3$

$a_1 == a_3$ $b_1 == b_3$

∴ x == z

Problem Two (Eq Laws: A Bad Instance)

Now consider the following instance of the Eq type class:

```
data BadPair a = B a a

instance Eq a => Eq (BadPair a) where
  -- (==) :: BadPair a -> BadPair a -> Bool
  (B x y) == (B x' y') = x == y' && y == x'
```

Note: If you want to prove that this instance does NOT satisfy a law, it suffices to provide a counter-example, i.e., a concrete instance which does not satisfy the law.

You could, for example, consider instances of `(BadPair Integer Integer)` when looking for counter-examples.

(a) Does this instance satisfy Reflexivity? Circle one: Yes [✓ No]

Now prove your answer:

x = BadPair a b   example : Bad Pair 1 2

$1 \neq 2$ and $2 \neq 1$

$x \neq x$

(b) Does this instance satisfy Symmetry? Circle one: [✓ Yes] No

Now prove your answer:

x = Bad Pair a_1 b_1 y = Bad Pair a_2 b_2

if x == y, then $a_1 == b_2 \wedge b_1 == a_2$

$a_2 == b_1 \wedge b_2 = a_1$

$\therefore$ y == x

(c) Does this instance satisfy Transitivity? Circle one: Yes [✓ No]

Now prove your answer:

x = Bad Pair 1 2   y = Bad Pair 2 1   z = Bad Pair 1 2

x == y because $1 == 1 \wedge 2 == 2$, y == z because $2 == 2 \wedge 1 == 1$

$x \neq z$ because $1 \neq 2$, $2 \neq 1$

Problem Three (Functor Laws: A Good Instance)

The laws for Functors are as follows, which should hold for any instance `T` of the Functor type class; the instance declaration for `T` must define an implementation of `fmap`. In the following, `f` and `g` are the functions being mapped over an instance `x` of `T`.

First Law: `∀x. fmap id x = id x` (where `id = \x -> x`)

Second Law: `∀f,g. (fmap f) . (fmap g) = fmap (f . g)`

An alternate form of the second law is perhaps easier to work with:

Second Law′ `∀x,f,g. (fmap f (fmap g x)) = fmap (f . g) x`

Now consider the following instance of the Functor type class:

```
data Pair a = P a a

instance Functor Pair where
 -- fmap :: (a -> b) -> Pair a -> Pair b
 fmap f (P x y) = P (f x) (f y)
```

(a) Prove that this satisfies the First Law.

fmap id (P x y) = P (id x) (id y) = P x y

(b) Prove that this satisfies the Second Law (note that you will have to consider *arbitrary* functions `f` and `g` of the appropriate types— because of the universal quantifier on these you can not simply pick two particular functions and show that the law works in those cases: what about all the other possible functions?).

fmap f (fmap g [illegible] (P x y)) = fmap f (P (g x) (g y))

= ~~fmap~~ P (f (g x)) (f (g y)) = P ((f . g) x) ((f.g) y)

= fmap (f.g) (Pair x y)

Problem Four (Functor Laws: A Bad Instance)

Consider the following instance of the Functor type class:

```
data BadList a = BNil | BCons a (BadList a)

instance Functor BadList where
  -- fmap :: (a -> b) -> BadList a -> BadList b
  fmap _ BNil = BNil
  fmap f (BCons x xs) = (BCons (f x) BNil)
```

(a) Does this instance satisfy the First Law? Circle one: Yes [✓ No]

Now prove your answer:

fmap id (BCons x xs) = BCons (~~f~~ id x) BNil

= BCons x BNil

≠ BCon x xs

(b) Does this instance satisfy the Second Law? Circle one: [✓ Yes] No

Now prove your answer:

fmap f (fmap g (BCons x xs)) = fmap f (BCons (g x) BNil)

= BCons (f (g x)) BNil

= BCons (f . g) x BNil

= fmap (f . g) (BCons x xs')