CS2030S Recitation Problem Set 8

Monad Laws

A monad is a *structure* with at least two methods (of, flatMap) obeying three laws:

1. Left Identity Law
    - $\forall x, f : \text{Monad.of}(x).\text{flatMap}(y \to \text{f(y)}) \equiv f(x)$
2. Right Identity Law
    - $\forall \text{monad} : \text{monad.flatMap}(x \to \text{Monad.of}(x)) \equiv \text{monad}$
3. Associative Law
    - $\forall \text{monad}, f, g : \text{monad.flatMap}(x \to f(x)).\text{flatMap}(y \to g(y))$
      $\equiv \text{monad.flatMap}(x \to f(x).\text{flatMap}(y \to g(y)))$

Functor Laws

A functor is a *structure* with at least 2 methods (of, map) obeying two laws:

1. Identity Morphism (basically mapping identity fn gives you the same functor)
   - $\forall \text{functor} : \text{functor.map}(x \to x)$
     $\equiv \text{functor}$
2. Composition morphism (any 2 maps is the same as 1 map with applying both function)
   - $\forall \text{functor}, f, g : \text{functor.map}(x \to f(x)).\text{map}(y \to g(y))$
     $\equiv \text{functor}, f, g : \text{functor.map}(x \to g(f(x)))$

Question 1a

Complete the implementation of map using only `flatMap` so that the resulting `Monad<T>` satisfies the functor laws.

- Need the identity and composition morphisms.

```
public <R> Monad<R> map(Transformer<? super T, ? extends R> f) {
  return this.flatMap(XXX); // Need to satisfy Functor laws
}
```

Question 1a

```
public <R> Monad<R> map(Transformer<? super T, ? extends R> f) {
  return this.flatMap(XXX); // Need to satisfy Functor laws
}
```

- Notice that $f : T \to R$
- What type should XXX be?
  - $\text{XXX} : T \to \text{Monad}{<}R{>}$
  - How can I use f to produce XXX?
  - XXX = `x -> Monad.of(f.transform(x))`
  - Remember `f.transform(x)` $\equiv$ `f(x)`

Question 1b

Prove that composition is preserved.

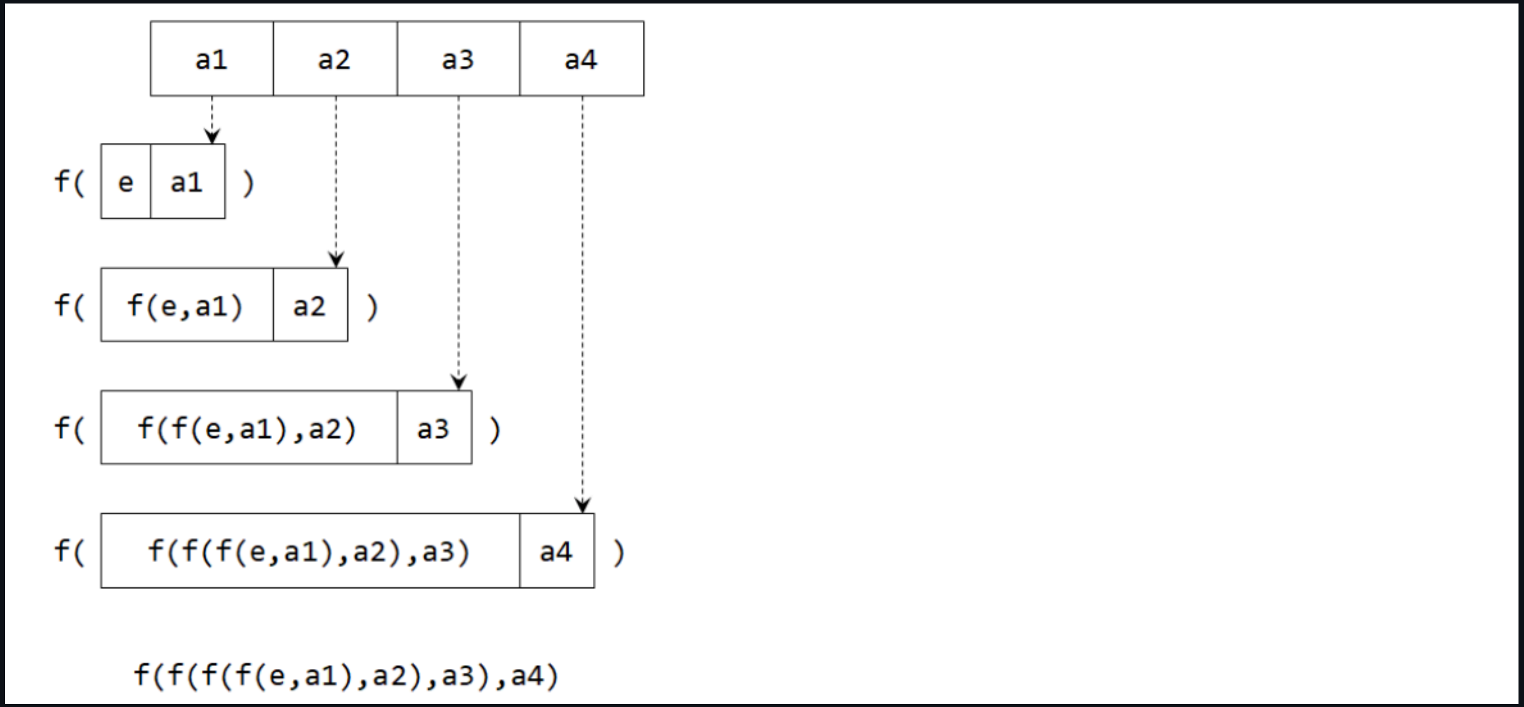
- A: `m.map(x -> f(x)).map(x -> g(x))` $\equiv$ `m.flatMap(x -> Monad.of(f(x))).flatMap(x -> Monad.of(g(x)))` by implementation
- B: $\equiv$ `m.flatMap(x -> Monad.of(f(x)).flatMap(x -> Monad.of(g(x))))` by associative law.
- C: $\equiv$ `m.flatMap(x -> Monad.of(g(f(x))))` by left identity law.

Sequential, Concurrent, and Parallel

- Sequential
  - Do things in order on one thread
- Concurrent
  - Do things in order one at a time but over different threads
- Parallel
  - Actually doing things at the same time

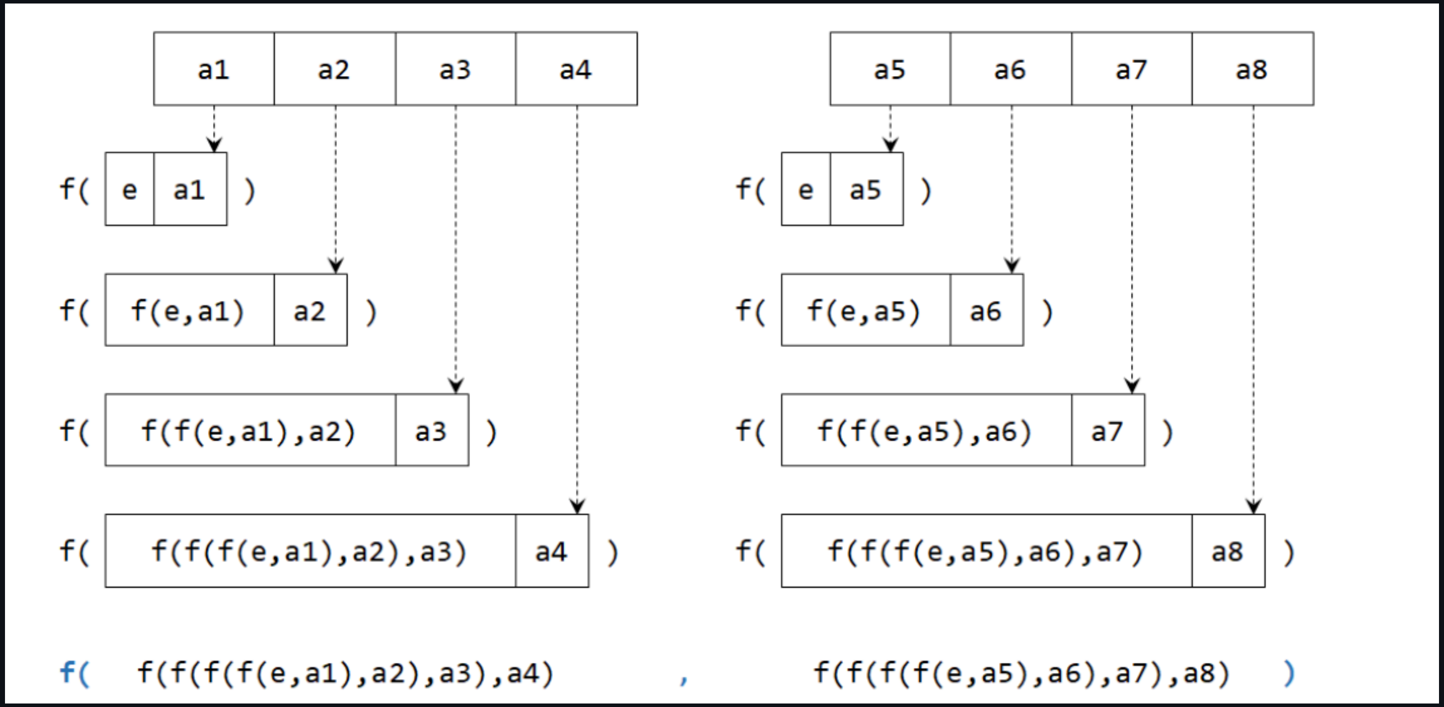
Reduce Sequential

```
T reduce(T e, BinaryOperator<T> f)
```



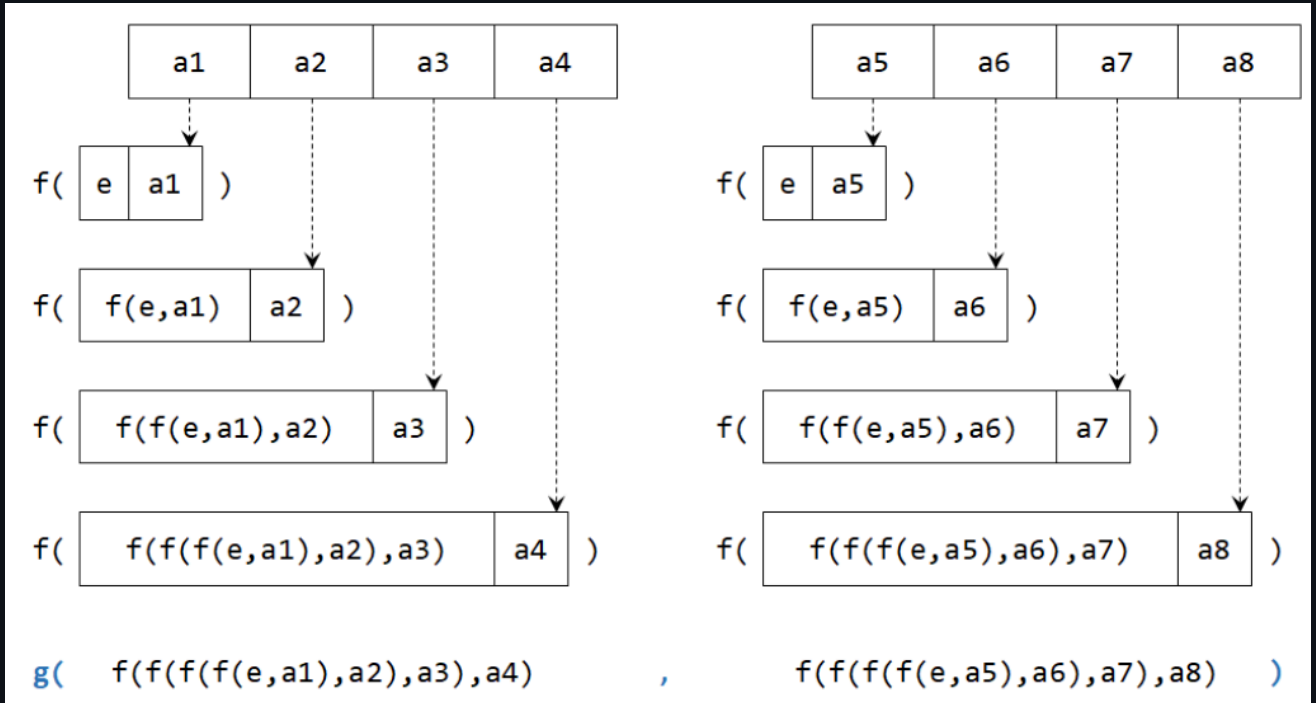
Reduce Parallel

```
T reduce(T e, BinaryOperator<T> f)
```



Reduce Parallel

```
<U> U reduce(U e, BiFunction<U,? super T,U> f, BinaryOperator<U> g)
```



Question 2a and 2b

What is the return value?

```
Stream.of(1, 2, 3, 4)
    .reduce(0, (a, x) -> (2 * a) + x, (a1, a2) -> a1 + a2);
```

```
Stream.of(1, 2, 3, 4)
    .parallel()
    .reduce(0, (a, x) -> (2 * a) + x, (a1, a2) -> a1 + a2);
```

Explain why there are differences

Reason

The accumulator is not associative

- If associative, $f(f(a,b),c) = f(a,f(b,c))$
- Future Brian will show you on the white board why it's not.

Write `estimatePi` using Stream

- Does parallelisation speed it up?
  - Show code
  - Overhead of creating new threads