Seminar 2.

COMP50001: Algorithm Design & Analysis

Sheet 2 (Week 3)

Exercise 2.1

Find a binary operation $(\diamond) :: (a \to a) \to (a \to a) \to (a \to a)$ and an element $\epsilon :: a \to a$ such that the set of functions of type $a \to a$ with $\diamond$ and ϵ forms a monoid.

Exercise 2.2

Given any two monoids $(M_1, \diamond_1, \epsilon_1)$ and $(M_2, \diamond_2, \epsilon_2)$, a *monoid homomorphism* from M_1 to M_2 is a function $h :: M_1 \to M_2$ such that

$$h\ (x \diamond_1 y) = (h\ x) \diamond_2 (h\ y)$$
$$h\ \epsilon_1 = \epsilon_2$$

Give three monoid homomorphisms from $([Int], +\!\!+, [\,])$ to $(Int, +, 0)$.

Exercise 2.3

Calculate the asymptotic time complexity of *concatl xs* below in terms of n and m where *xs* contains n lists, each containing m elements.

$$concatl :: [[a]] \to [a]$$
$$concatl = foldl\ (+\!\!+)\ [\,]$$

See lecture or below...

Exercise 2.4

The *List* type class is shown in Figure 2.4. Complete the specification of the *List* type class by providing a default implementation for all the operations other than *fromList* and *toList*. ✓

Exercise 2.5

Implement an instance of *List* using standard lists $[a]$ without using functions from the *Prelude* other than the list constructors, and give the time complexities of each operation. ✓

Exercise 2.6

Implement an instance of *List* using the following *Tree* type:

$$\textbf{data}\ Tree\ a = Tip \mid Leaf\ a \mid Fork\ (Tree\ a)\ (Tree\ a)$$

Ensure that the worst case complexity of $(+\!\!+)$ is $O(1)$. What is the worst case complexity of *head*?

```
class List list where
  fromList :: [a] → list a
  toList :: list a → [a]
  normalize :: list a → list a

  empty :: list a
  single :: a → list a
  cons :: a → list a → list a
  snoc :: list a → a → list a

  head :: list a → a
  tail :: list a → list a
  init :: list a → list a
  last :: list a → a

  isEmpty :: list a → Bool
  isSingle :: list a → Bool
  length :: list a → Int
  (++) :: list a → list a → list a
  (!!) :: list a → Int → a
```

Figure 1: List class definition

```
snoc :: [a] → a → [a]
snoc [] y = [y]
snoc (x:xs) y =
  x : snoc xs y
```

Exercise 2.7

Define an instance of *List* using *DList* below, and give the complexities of all operations in terms of the length of the input list (assume all *DList* arguments to functions are built using the operations in *List*).

~~newtype~~ *DList a* = *DList* ($[a] \to [a]$)

Hint: *fromList xs* = *DList* ($xs{+\!\!+}$). Consider carefully whether the time complexity is affected by strict or lazy evaluation.

Exercise 2.8

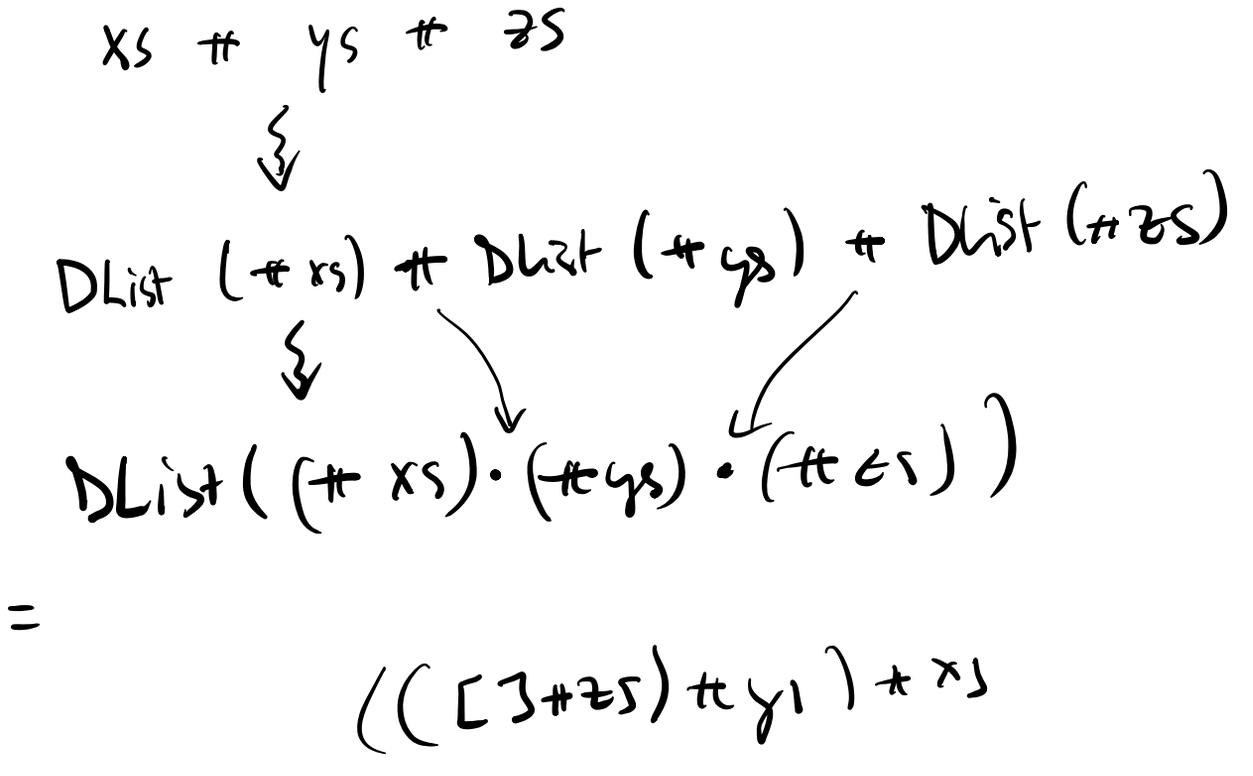
Explain why the following implementation of fromList is undesirable in the last exercise:

fromList xs = *DList* ($+\!\!+xs$)

Exercise 2.9

Prove or disprove the following assertions for the *DList* instance of *List* from Exercise (2.7).

1. *fromList* (*toList dxs*) = *dxs* for any *dxs* :: *DList a*.
2. *toList* (*fromList xs*) = *xs* for any *xs* :: $[a]$.



Exercise 2.1

Find a binary operation $(\diamond) :: (a \to a) \to (a \to a) \to (a \to a)$ and an element $\epsilon :: a \to a$ such that the set of functions of type $a \to a$ with $\diamond$ and ϵ forms a monoid.

The operation is $(\cdot) :: (b \to c) \to (a \to b) \to (a \to c)$

$(\cdot) :: (a \to w) \to (a \to c) \to (a \to a)$

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

Associativity:

$$f \cdot (g \cdot h) = (f \cdot g) \cdot h$$

$\Leftrightarrow$

$$(f \cdot (g \cdot h))\ x = ((f \cdot g) \cdot h)\ x$$

$\Leftrightarrow$

...

$\Leftrightarrow$

$$f\ (g\ (h\ x)) = f\ (g\ (h\ x))$$

Units:

$$(id \cdot g)\ x = id\ (g\ x) = g\ x \qquad (f \cdot id)\ x = f\ (id\ x) = f\ x$$

Exercise 2.2

Given any two monoids $(M_1, \diamond_1, \epsilon_1)$ and $(M_2, \diamond_2, \epsilon_2)$, a *monoid homomorphism* from M_1 to M_2 is a function $h :: M_1 \to M_2$ such that

$$h\ (x \diamond_1 y) = (h\ x) \diamond_2 (h\ y)$$
$$h\ \epsilon_1 = \epsilon_2$$

Give three monoid homomorphisms from $([Int], +\!\!+, [\,])$ to $(Int, +, 0)$.

h_1 = foldr (+) 0 = sum
h_2 = length (x n)
h_3 = const 0

const 0 (xs ++ ys) = 0
=
0 + 0
=
const 0 xs + const 0 ys

T(k, n, m) ← length of xs ∈ xss
↑ length xss
↑ length ys

foldl (++) ys xss

foldl (++) ys [] =

k T(k, 0, m) = 1

foldl f k [] = k

```
foldl (⊕) ys (xs : xss) =
    foldl (⊕) (ys ++ xs) xss
```

$$* \quad T(k, n, m) = k + T(k+m, n-1, m)$$

```
concatl xss = foldl (++) [] xss
```

$$
\begin{aligned}
& T(0, n, m) \\
= \; & 0 + T(m, n-1, m) \\
= \; & 0 + m + T(2m, n-2, m) \\
= \; & 0 + m + 2m + T(3n, n-3, m) \\
= \; & \cdots \\
= \; & \sum_{k=0}^{n-1} k \times m \; + T(nm, n-m, m)
\end{aligned}
$$

$$\in O(n^2 m).$$

Exercise 2.4

The *List* type class is shown in Figure 2.4. Complete the specification of the *List* type class by providing a default implementation for all the operations other than *fromList* and *toList*.

```
class List list where
  fromList :: [a] → list a
  toList :: list a → [a]
  normalize :: list a → list a

  empty :: list a
  single :: a → list a
  cons :: a → list a → list a
  snoc :: list a → a → list a

  head :: list a → a
  tail :: list a → list a
  init :: list a → list a
  last :: list a → a

  isEmpty :: list a → Bool
  isSingle :: list a → Bool
  length :: list a → Int
  (++) :: list a → list a → list a
  (!!) :: list a → Int → a
```

```
empty = fromList []

Prelude.
head xs = head (toList xs)
          xs
          list a

cons x xs = fromList (x : toList xs)
```

Exercise 2.6

Implement an instance of *List* using the following *Tree* type:

data *Tree a* = *Tip* | *Leaf a* | *Fork* (*Tree a*) (*Tree a*)

Ensure that the worst case complexity of $(++)$ is $O(1)$. What is the worst case complexity of *head*?

```
(++) :: Tree a → Tree a → Tree a

~~Tip ++ Leaf x = Leaf x~~

Tip ++ ys  = ys
xs ++ Tip  = xs
xs ++ ys   = Fork xs ys
```

head Tip = error "head: Tip"
head (Leaf x) = x
head t = head' (normalize t)

Exercise 2.9

Prove or disprove the following assertions for the *DList* instance of *List* from Exercise (2.7).

1. *fromList* (*toList dxs*) = *dxs* for any *dxs* :: *DList a*.
2. *toList* (*fromList xs*) = *xs* for any *xs* :: [*a*].

Consider
DList (reverse) :: DList a
[a] → [a]

fromList (toList (DList reverse))
= fromList (reverse [])
= fromList []
= DList (λxs → xs ++ [])
=

DList (λxs → xs)

=

DList id

toList (fromList xs) $\stackrel{?}{=}$ xs.

=

toList (DList (λys → xs ++ ys))

=

(λys → xs ++ ys) []

=

xs ++ []

=

xs