LECTURE 5: Abstraction

"We must beware of needless innovation, especially when guided by logic"

— Winston Churchill, 1942

A monoid is a triple $(M, \diamond, \varepsilon)$ with laws. We represent them with a class:

PS. $\diamond$ is written <> in ASCII

```
> class Monoid m where
>     (<>) :: m -> m -> m
>     mempty :: m
```

(ε ↑)

such that associativity & unit laws hold.

```
> instance Monoid Int where
>     (<>) = (+)
>     mempty = 0
```

but what about $(\mathbb{Z}, (*), 1)$?

```
> newtype MultInt = MultInt Int
```

($\cong$)

MultInt ≅ Int

```
> instance Monoid MultInt where
>     (<>) = (*)
>     mempty = 1
```

We can abstract lists to a class

```
> class List list where
>     empty :: list a
>     cons  :: a -> list a -> list a
>     snoc  :: list a -> a -> list a
>     head  :: list a -> a
>     tail  :: list a -> list a
>     (++)  :: list a -> list a -> list a
>     ...
```

[1,2,3] :: [Int]
[] [2,3] :: [Int]
[3] :: [Int]
[] :: [a]

head (cons x xs) = x

conc (head xs) (tail xs) = xs

There are two more critical functions:

```
> toList   :: list a -> [a]
> fromList :: [a] -> list a
```

abstract ↑ ↓ concrete

The critical law is:

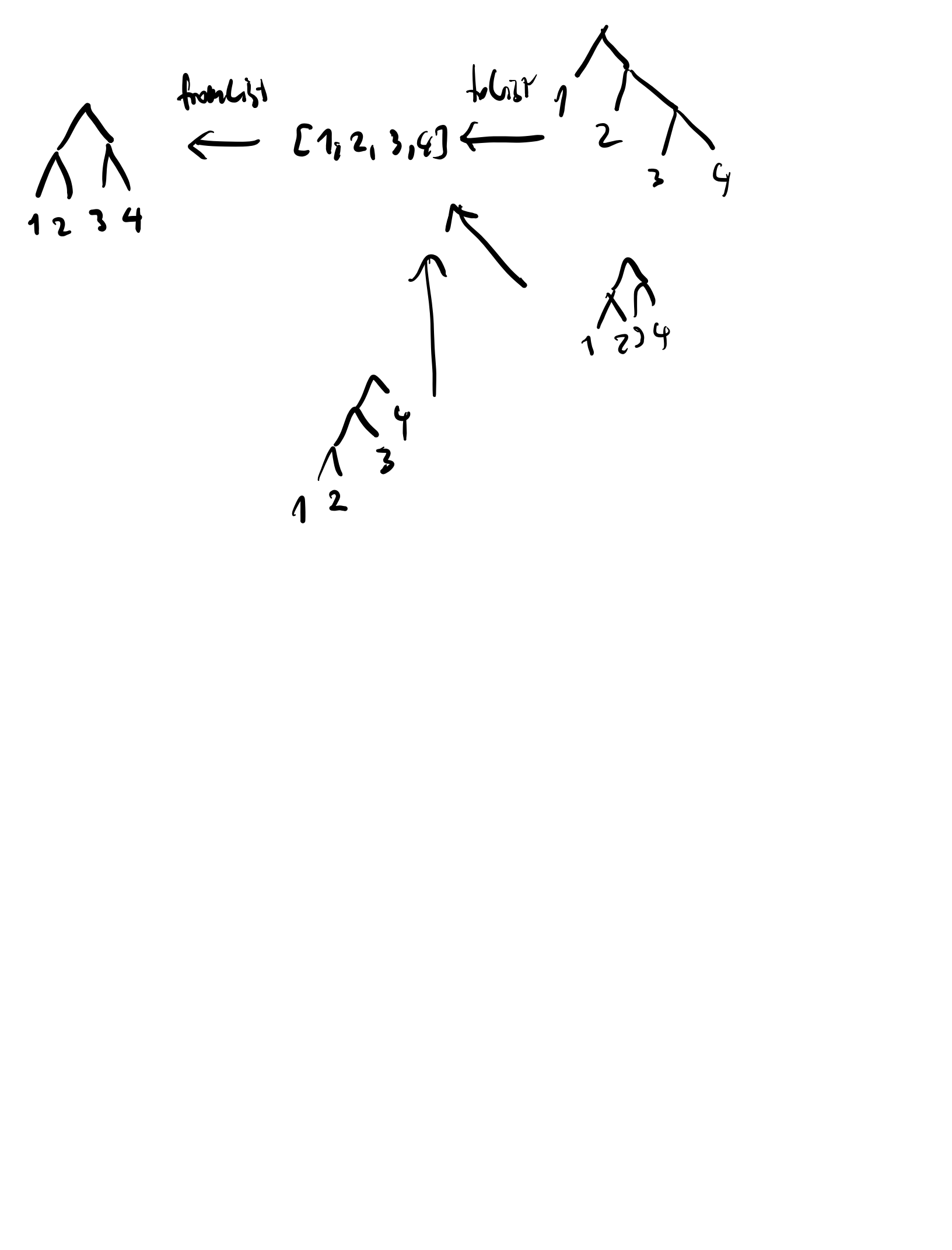
$$id = toList \cdot fromList$$

But this is not true in general:

$$id = fromList \cdot toList$$

but it's still useful:

$$normalise :: fromList \cdot toList$$


fromList
toList
[1, 2, 3, 4]
1 2 3 4
1
2
3
4
1 2 4
1 2
3
4