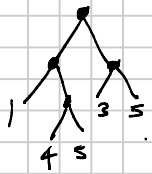
CS 558 — 19 September 2024

Note Title 2024-09-19

Binary trees




```
data Tree a =
      Leaf a
    | Node (Tree a) (Tree a)
```

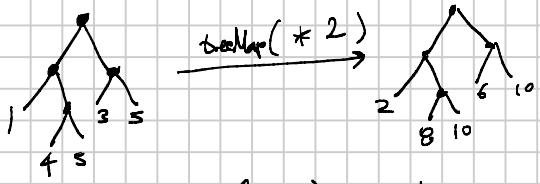
left subtree, right subtree

```
treeExample :: Tree Int
treeExample = Node (Node (Leaf 1) (Node (Leaf 4) (Leaf 5)))
                   (Node (Leaf 3) (Leaf 5))

Leaf :: a -> Tree a
Node :: Tree a -> Tree a -> Tree a
```

```
sum :: Num a => Tree a -> a
sum (Leaf x) = x
sum (Node t1 t2) = sum t1 + sum t2
```

```
treeMap :: (a -> b) -> Tree a -> Tree b
treeMap f (Leaf x) = Leaf (f x)
treeMap f (Node t1 t2) = Node (treeMap f t1) (treeMap f t2)
```




```
treeFold :: (a -> b) -> (b -> b -> b) -> Tree a -> b
```

fLeaf: a function describing how to turn a value in a Leaf to get a result of type b

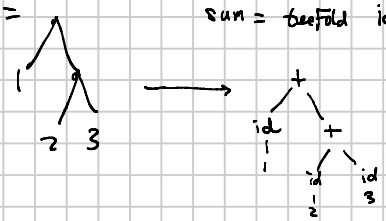
fNode: how to combine the results of the two recursive calls for the two subtrees into a single result

Tree a: input tree

b: final result

```
treeFold fLeaf fNode (Leaf x) = fLeaf x
treeFold fLeaf fNode (Node t1 t2) =
    fNode (treeFold fLeaf fNode t1)
          (treeFold fLeaf fNode t2)
```

```
t :: Tree Int
t =
```




```
sum = treeFold id (+)

(+) (id 1) ((+) (id 2) (id 3))
```

Express treeMap using treeFold:

```
treeMap f = treeFold (Leaf . f) Node
```

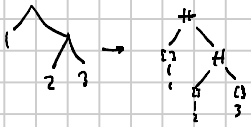
f :: a -> c; t :: Tree a; (Leaf . f) :: a -> b; Node :: b -> b -> b; b = Tree c; t :: Tree c

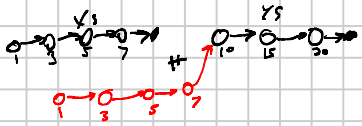
Express: enumeration of all elements using treeFold; depth-first:

```
Tree a -> [a]
```

→ [1,2,3]

```
enumerate :: Tree a → [a]        a → [a]      [a] → [a] → [a]
enumerate = treeFold (\n → [n]) (++)
```





Proving properties of functions on trees:

To prove $\forall t :: \text{Tree } a,\ P(t)$

(for all finite trees of type Tree a)

we can prove:

① Base case

$\forall x :: a.\ P(\text{Leaf } x)$

② Inductive case.

$\forall t1 :: \text{Tree } a.\ \forall t2 :: \text{Tree } a$

$(P(t1) \wedge P(t2)) \Rightarrow P(\text{Node } t1\ t2)$