# Title

Brian Kuhns

April 2019

```
module BinTree where
import Control.Comonad
```

This bin-tree module is writen to support derivation histories. For this reason each Node has exactly two children and the leavs which represent givens have no children.

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

The functor instance is pretty standard, the function is mapped over each element in the tree.

```
instance Functor BinTree where
  fmap f (Leaf x) = Leaf $ f x
  fmap f (Node x l r) = Node (f x) (fmap f l) (fmap f r)
```

For the Comonad instance. Extract returns the root of the tree. Duplicate replaces each poin in the tree with the subtree of which it is the root.

The comonad laws are:

- extend extract = id
- extract . extend f = f
- extend f . extend g = extend (f . extend g)

```
instance Comonad BinTree where
  extract (Node x _ _) = x
  extract (Leaf x )    = x
  duplicate t@(Leaf _) = Leaf t
  duplicate t@(Node _ l r)  = Node t (duplicate l) (duplicate r)
```

The Foldable instance is also pretty standard, It goest left children node right children

```
instance Foldable BinTree where
  foldMap f (Leaf x) = f x
  foldMap f (Node x l r) = (foldMap f l) <> (f x) <> (foldMap f r)
```

This function gets all the leaves of the tree as a list.

```
leaves :: BinTree a -> [a]
leaves (Leaf x) = [x]
leaves (Node _ l r) = leaves l ++ leaves r
```

The show instance shows the tree as structure with indentation putting each element on a different line. The leaves are prefixed with G to denote givens, and the nodes with D to denote derived.

```
instance Show a => Show (BinTree a) where
  show (Leaf n) = "G " ++ show n
  show (Node n h1 h2) = "D " ++ (show n) ++ "\n" ++  (indent . show $ h1) ++ "\n" +
(indent . show $ h2)
    where
      indent = init . unlines . map ("  " ++) . lines
```