Problem: label each node with its position in the tree’s postorder traversal

postOrderLabel :: BTree a --> BTree a
i = 0

def post_order_label(tree):
    if tree.is_empty():
        return
    else:
        post_order_label(tree.left)
        post_order_label(tree.right)
        tree.root.label = i
    i = i + 1
data State s a = State (s -> (a, s))

get :: State s s
get = State (\s -> (s, s))

put :: s -> State s ()
put x = State (\s -> (()), x))

runState :: State s a -> (s -> (a, s))
runState (State f) = f
Today: the last two “famous” monads
Non-deterministic choice revisited
instance Monad [] where
    return :: a -> [a]
    (>>=) :: [a] -> (a -> [b]) -> [b]
concatMap: combining choices

(+ (\(< 1 2 3\)) (\(< 4 5 6\)))

“for each choice from [1,2,3], combine it with each of [4,5,6], then concatenate all the results”
Demo!
Notes

This is how Haskell’s list comprehensions are implemented
Notes

Very similar to Racket’s choice operator –<, except doesn’t capture continuations.
Talking to the outside world: IO

“If a program runs without any side effects, does it make a sound?”
Key idea: treat the outside world as some big mutable State
getLine :: World -> (String, World)
print :: String -> World -> (() , World)
open :: String -> World -> (FILE *, World)
read :: FILE * -> World -> (String, World)
getMouseClick :: World -> (Event, World)
run :: String -> World -> (PID, World)
checkFacebook :: World -> (😐, World)
submitWork :: Assignment ->
    World -> (★, World)
State World a  <-- doesn’t exist
State World a  <-- doesn’t exist
IO a          <-- the real deal
\textbf{I0} \textbf{a} represents an I/O action that produces a value of type \textbf{a}. 
Demos!