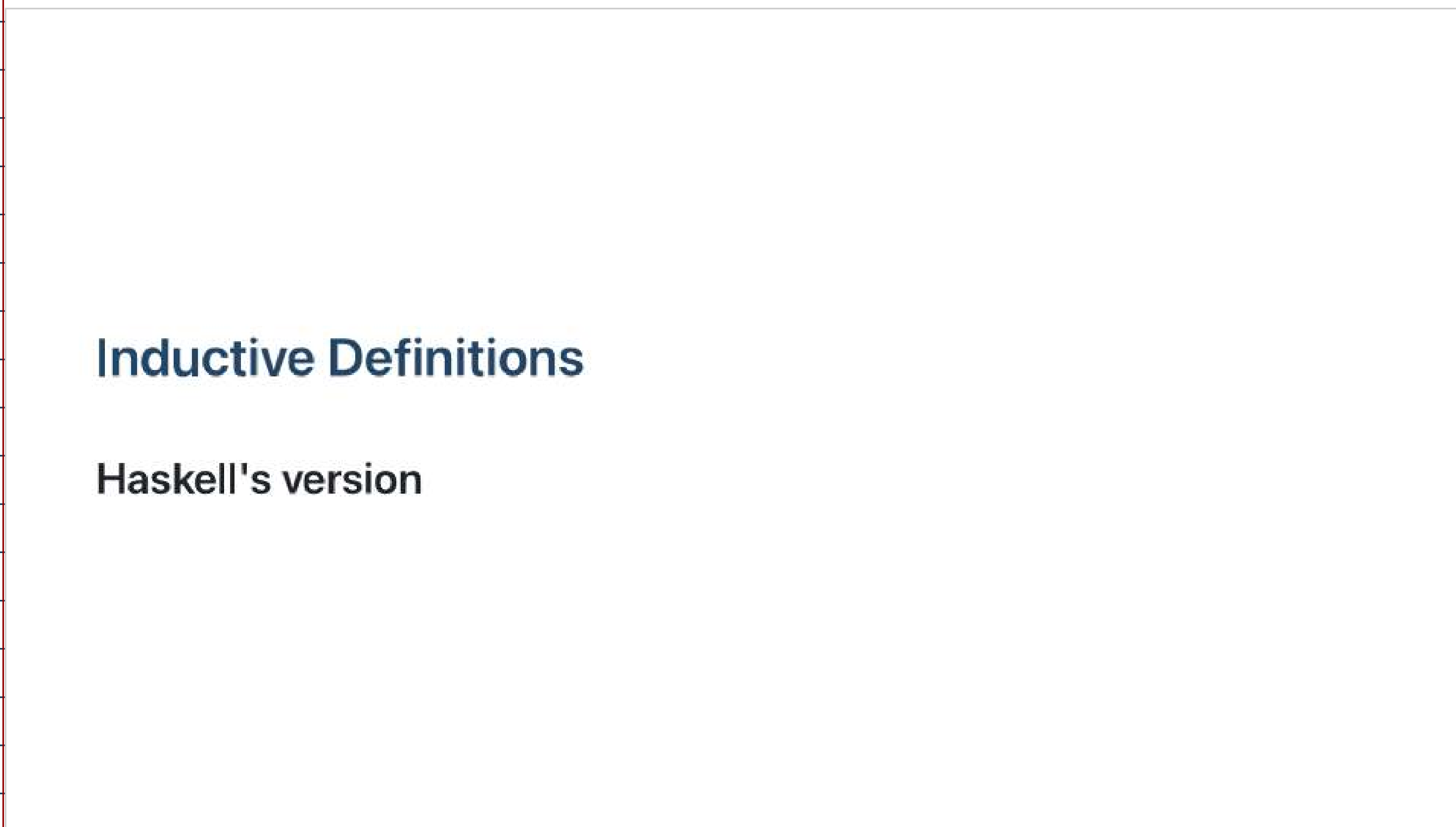
3 - Inductive Definitions (Haskell's version)

September 11, 2024 8:30 AM

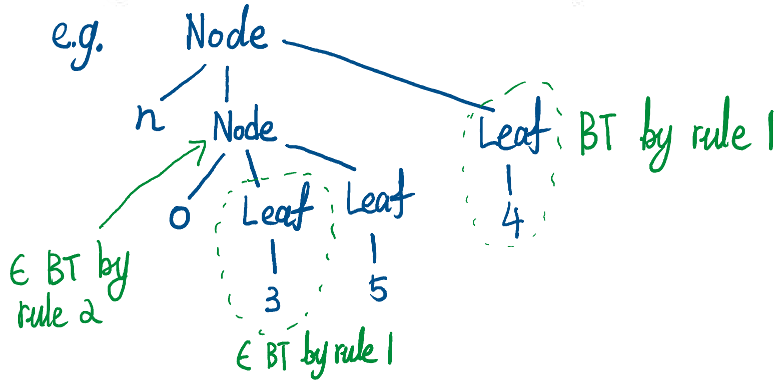


Example: binary trees [botched by prof last lecture]

Inductively define the set `BT` of binary trees with integer "info" as follows.

```
    n ∈ Z          n ∈ Z    t1 ∈ BT    t2 ∈ BT
-------------   ---------------------------
 Leaf n ∈ BT          Node n t1 t2 ∈ BT
```

Example tree with derivation and tree picture:



Another view of incr

Last time, we had a pseudo-Python function for adding *1* to the numbers in a BT.

```
incr(t):
  if t is Leaf n:
    n+1
  else t is Node n t1 t2:
    Node (n+1) (incr t1) (incr t2)   # recursive calls on smaller trees
```

Rewriting,

```
if t is Leaf n then incr(t) = Leaf (n+1)
if t is Node n t1 t2 then incr(t) = Node (n+1) (incr t1) (incr t2)
```

Valid Haskell code

```
incr(Leaf n) = Leaf (n+1)
incr(Node n t1 t2) = Node (n+1) (incr t1) (incr t2)
```

"pattern-matching" + recursion

We've arrived at Haskell

Haskell directly supports inductive definitions, pattern-matching case analysis and recursion.

```
-- Type of binary trees
data BT = Leaf Int | Node Int BT BT
```

either Leaf with int
or Node with int BT BT

constructor, must be capitalized, variable must be lower case.

This means exactly the same thing as the corresponding inductive definition.
I.e. the Haskell type BT has exactly the values generated by the rules below.

```
                  n ∈ Int   ⇒  Leaf n ∈ BT
n ∈ Int, t1 ∈ BT, t2 ∈ BT   ⇒  Node n t1 t2 ∈ BT
```

rules

Evaluation details: example

actual program

Leaf

```
incr (Leaf n) = n+1
incr (Node n t1 t2) = Node (n+1) (incr t1) (incr t2)
```

Steps Haskell takes to evaluate `incr v` for some tree value `v`:

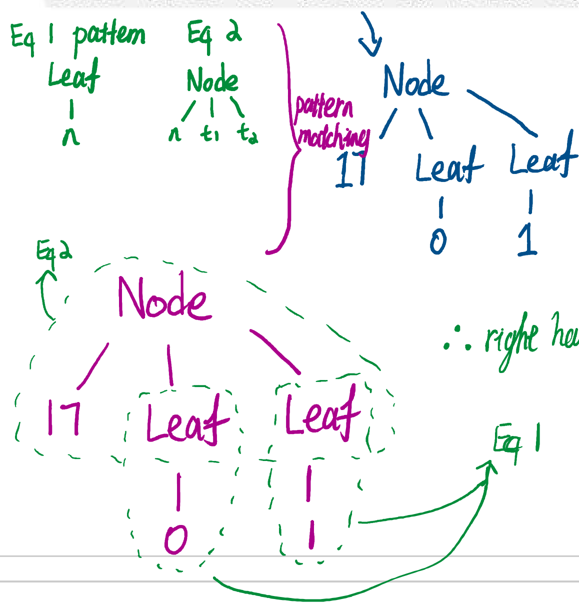
1. Find first equation whose lhs (left-hand-side) matches `v`.
2. Get the variable values from the match.
3. Plug the values in for the variables on the rhs and continue by evaluating the rhs.

E.g. evaluate `incr (Node 17 (Leaf 0) (Leaf 1))`.

Tree view of pattern-matching

```
(1) incr (Leaf n) = n+1
(2) incr (Node n t1 t2) = Node (n+1) (incr t1) (incr t2)
```

`incr (Node 17 (Leaf 0) (Leaf 1))`.



∴ right hand side (rhs) of equation 2:

Node (17+1) (incr (Leaf 0)) (incr (Leaf 1))

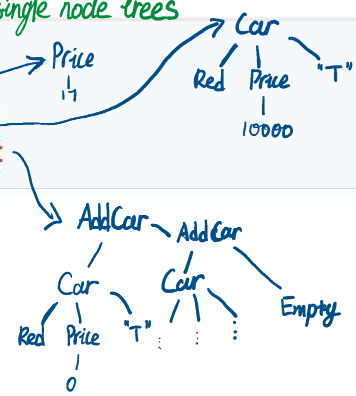
and keep computing to get:

Node (18) (Leaf 1) (Leaf 2)

Example: cars!

```
data Colour = Blue | Red | Yellow
data Price  = Price Int
data Car    = Car Colour Price String
data Fleet  = Empty | AddCar Car Fleet
```

single node trees



Exercises: see the accompanying Haskell file.

Draw a tree for the fleet example

```
car0 = Car Red (Price 60000) "Lincoln Juggernaut"
car1 = Car Yellow (Price 120000) "BMW Highsnoot"
car2 = Car Blue (Price 10000) "Fiat Roadkill"
fleet = AddCar car0 (AddCar car1 (AddCar car2 Empty))
```