From `Void` to `Pointer.Mu`

Guillaume Allais

SPLV St Andrews

July 24^{th}–28^{th} 2023

Me, Myself, and I

Who am I?

- Lecturer at the University of Strathclyde
- Idris and Agda core developer

What do I care about?

- Correct-by-construction method
- Efficient runtime representations
- Generic programming
- Partial Evaluation
- "Practical" dependently typed libraries
- Proof Automation

Plans for this week

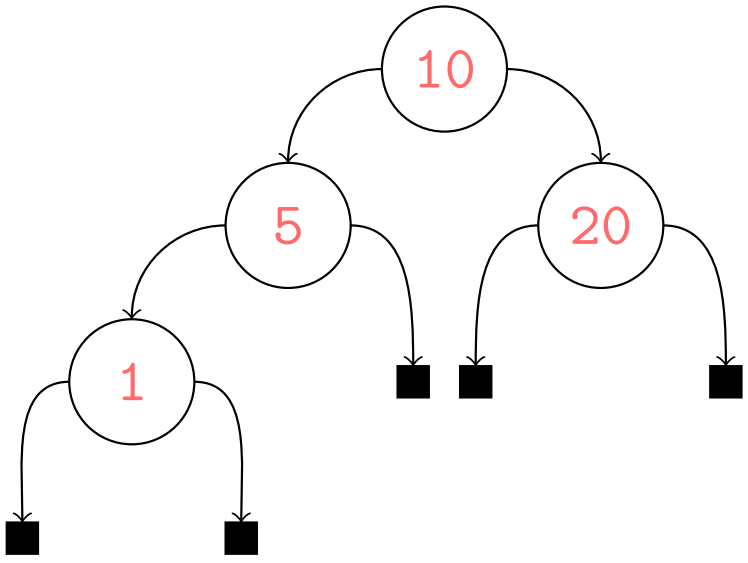
(WIP) Lecture Notes:
`https://gallais.github.io/teaching`

1. Motivation and Introduction to Idris 2
   - No prerequisites
   - Live programming
   - Key ideas, design patterns

2. Generic Programming
3. Programming Over Serialised Data

Motivation

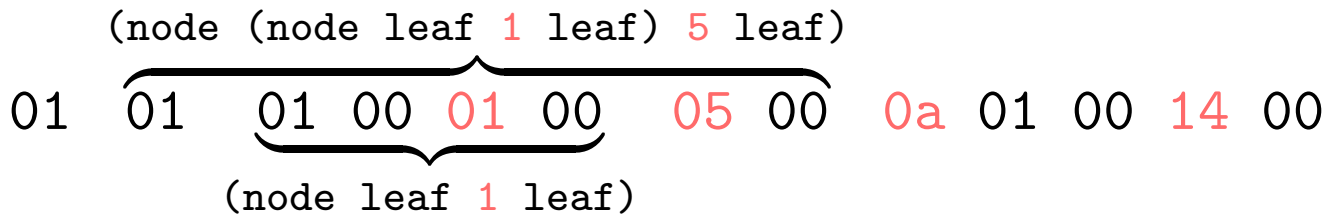
Trees and Pattern Matching




```
data Tree
  = Leaf
  | Node Tree Bits8 Tree

sum : Tree -> Nat
sum t = case t of
  Leaf => 0
  Node l b r =>
    let m = sum l
        n = sum r
    in (m + cast b + n)
```

Serialised Data and Pointer Manipulations



Serialised Data and Pointer Manipulations


```
1  int sumAt (uint8_t buf[], int *ptr) {
2    uint8_t tag = buf[*ptr]; (*ptr)++;
3    switch (tag) {
4      case 0: return 0;
5      case 1:
6        int m = sumAt(buf, ptr);
7        uint8_t b = buf[*ptr]; (*ptr)++;
8        int n = sumAt(buf, ptr);
9        return (m + (int) b + n);
10     default: exit(-1); }}
```

Seamless

```
sum : Data.Mu Tree -> Nat
sum t = case t of
  "Leaf" # _ => Z
  "Node" # l # b # r =>
    let m = sum l
        n = sum r
    in (m + cast b + n)
```

```
sum : Pointer.Mu Tree _ -> IO Nat
sum ptr = case !(view ptr) of
  "Leaf" # _ => pure Z
  "Node" # l # b # r =>
    do m <- sum l
       n <- sum r
       pure (m + cast b + n)
```

Correct

```
sum : Pointer.Mu Tree t ->
      IO (Singleton (Data.sum t))
sum ptr = case !(view ptr) of
  "Leaf" # _ => pure [| Z |]
  "Node" # l # b # r =>
    do m <- sum l
       n <- sum r
       pure [| [| m + [| cast b |] |] + n |]
```

Generic

```
fold : {cs : Data nm} -> (alg : Alg cs a) ->
       forall t. Pointer.Mu cs t ->
       IO (Singleton (Data.fold alg t))
```

Live programming