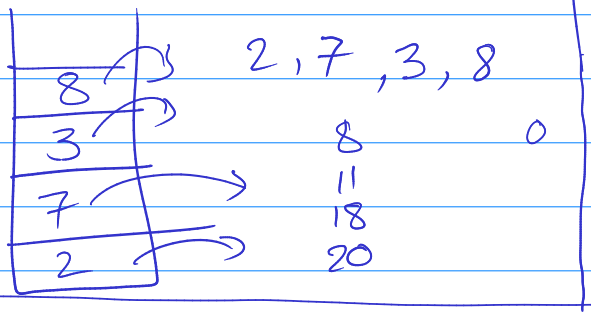
foldl ... (x::xs) = foldl ... (xs)

foldl . f . acc . list

```
{
acc = ___ ;
while (list != [])
 { acc ← f(acc, head(list));
   list ← tail(list);
 }
return acc;
}
```



foldr . f . idt . list

idt tail . list

(x::xs) = f . x . (foldr . f . id . xs)

```
foldr (fun*, idt, list)
{ stack ← [];
  while (list != [])
  { push (head(list));
    list ← tail(list);
  }
  t ← id;
  while (stack != []) { t ← f(pop(), t)
  }
  return t;
}
```

ctype [a] where

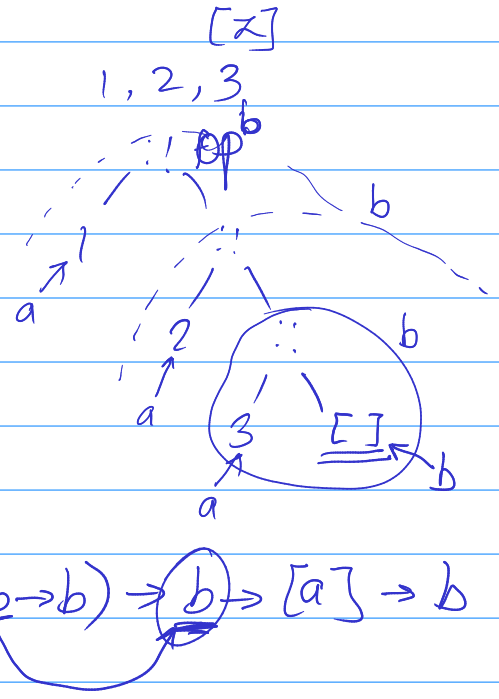
z ← [] : [a]

(::) : a → [a] → [a]

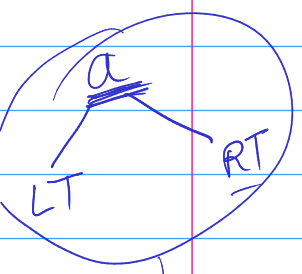
fn

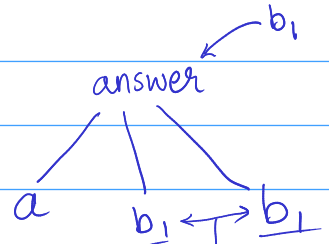
cons = foldr . (::) . []

foldl . f . acc



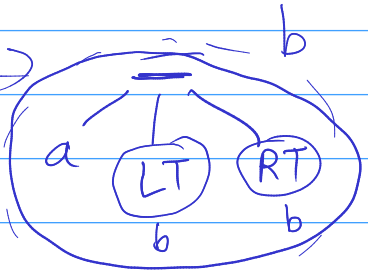
(a → b → b) → b → [a] → b





assumption : both are same types

$[a]$ $\$$ op $f : (a \to b \to b)$



$f : (a \to b \to b \to b)$

$(a \to b \to b)$ $(a \to a \to b)$

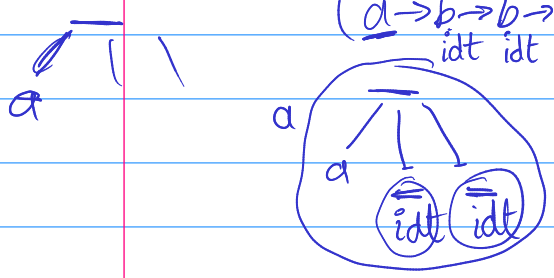
$(a \to a \to a)$

foldTree . f . __ . (tree)

foldTree : (f) → idt → retf → Tree.a

$(a \to b \to b \to b)$ → b → b → Tree.a

idt idt



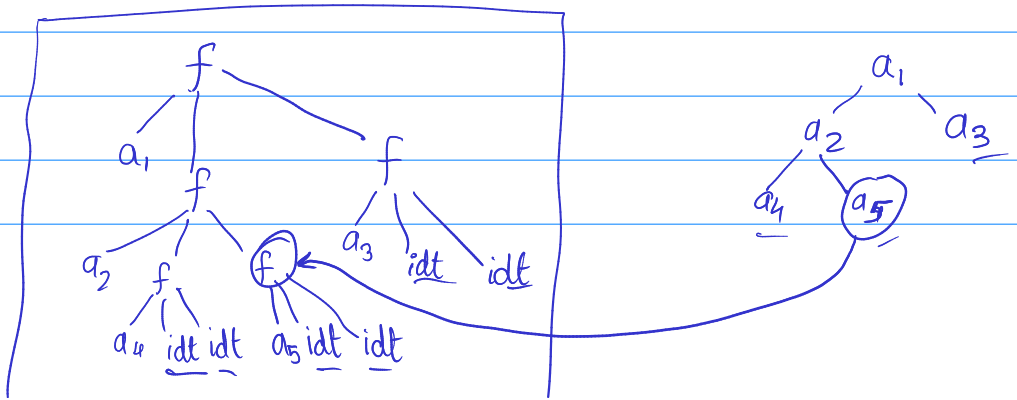
foldTree : $(a \to b \to b \to b)$ → b → Tree.a → b

foldTree . f . idt . (Lf . x) =

foldTree . f . idt . (Br . x . l . r) =

Lf . x

Br . x . Emt . Emt



ctype Tree.a where

Emt : Tree.a

Br : a → Tree.a → Tree.a → Tree.a