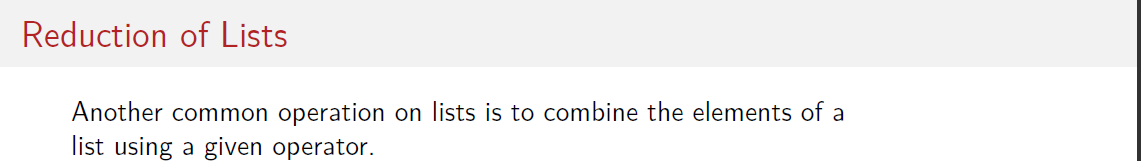
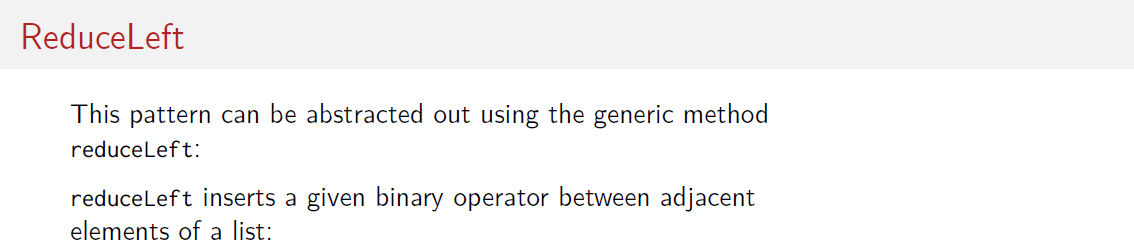
**Reduction of list**

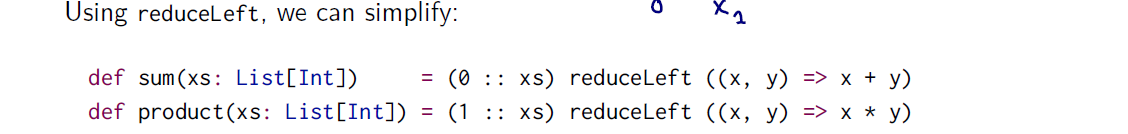
****

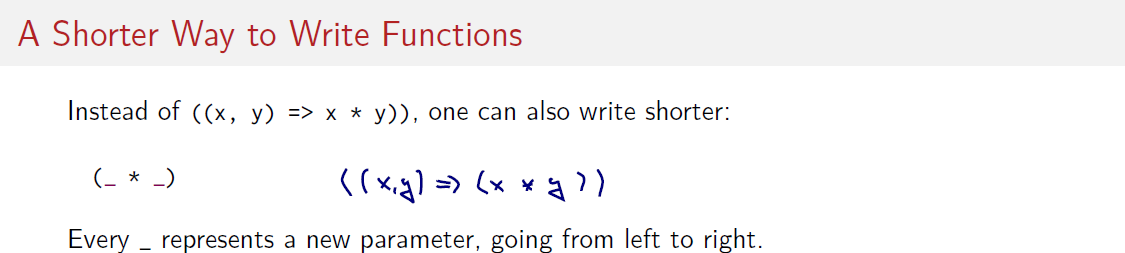
**For example sum () or product all elements**

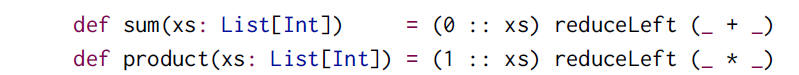
**sum()= 0+x1+….+xn**

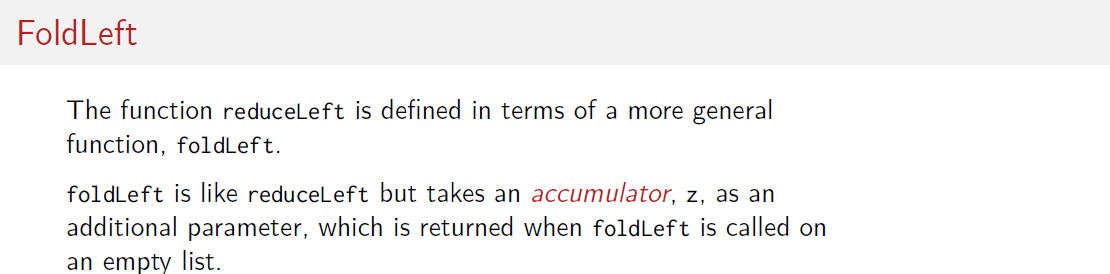
**product = 1\*x1.... +xn**

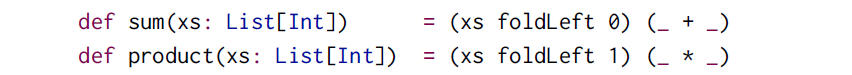


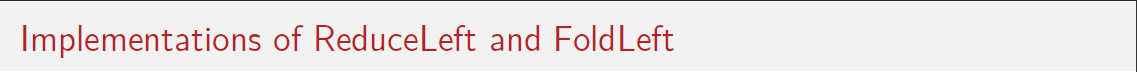


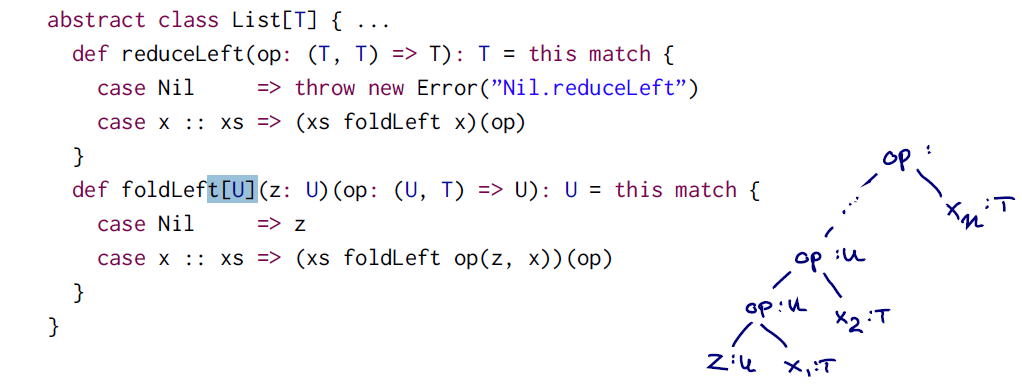






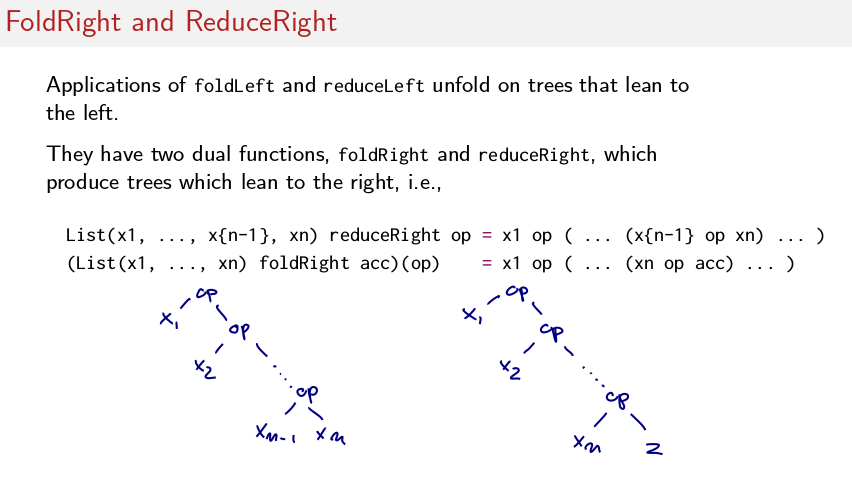






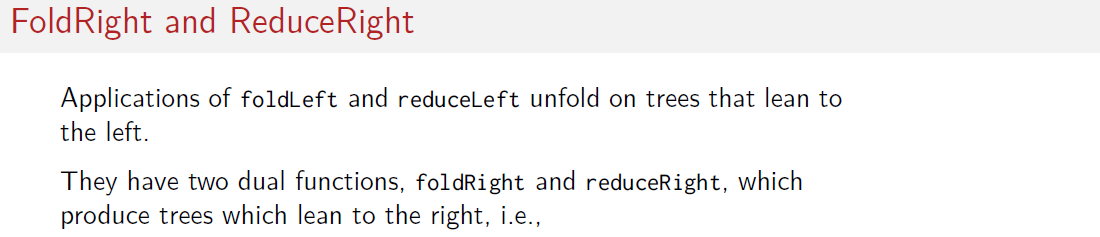
Note that in the second case ( x::xs ) in foldLeft the zero element “z” is always passed in the recursive calls. This is the accumulator.

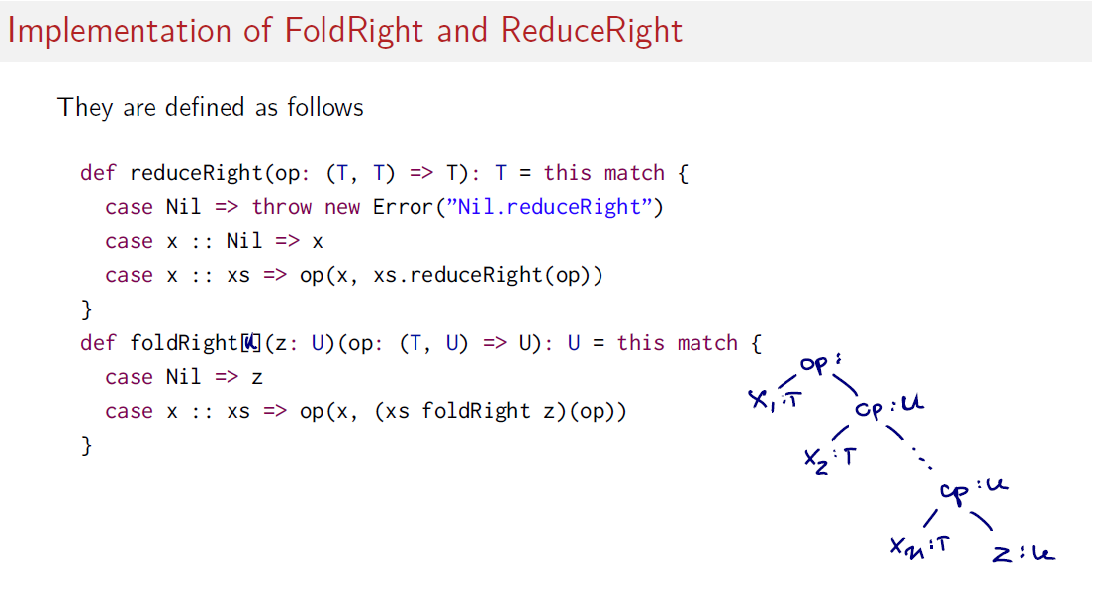
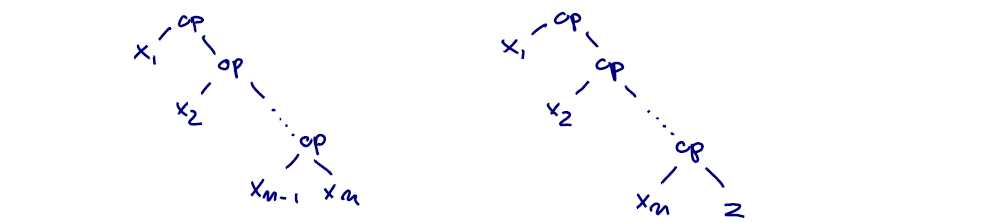
“xs foldLef op( z,x) (op)”. For that reason it is applied in each operation, better descried in the drawing above. It is a classic loop with an accumulator. Where op( z,x) return an result of type U then “xs.foldLef( result ) (op)”. On and On!!

Note that now the prentices goes to the right !

Left --> (1+2 )+3 from the beginning to the end.

Right --> 1+( 2+3) from the last element to beginning





1 in reduceRight: op( x, xs.reduceRight(op ) ) the operation of the first element followed by the result of a call of xs.reduceRight(op) of the second element “xs” recursively.

2 in foldRight() : op(x, ( xs foldRigh z) (op) ) means an operation that tales the header of the fist element and the result of applying foldRight

recursively to the tail of the list.

