

# Folding 101

# Simplified folding

foldl1

# - Visualise a list of elements

$[\alpha, \beta, \gamma, \delta]$

- Visualise a list of elements
- Remove outer square brackets

$\alpha$  /  $\beta$  /  $\gamma$  /  $\delta$

- Visualise a list of elements
- Remove outer square brackets
- Replace commas with a single binary operator

$$\alpha \times \beta \times \gamma \times \delta$$

- Visualise a list of elements
- Remove outer square brackets
- Replace commas with a single binary operator
- Evaluate expression

$\alpha \times \beta \times \gamma \times \delta$

$\Rightarrow \Sigma$

# Examples

- Summation - foldl1 (+)
- Logical And - foldl1 (&&)
- Bananas - foldl1 mash

# Visualise Elements (as when printed)

[1,2,3,4]

[True,True,False]

[Banana,Banana,Banana]



# Remove outer square Brackets

1,2,3,4

True,True,False

Banana,Banana,Banana

# Replace commas with an operation

1+2+3+4

True && True && False

Banana `mash` Banana `mash` Banana

# Evaluate (in this case left to right)

1+2+3+4

=> 10

True && True && False

=> False

Banana `mash` Banana `mash` Banana

=> MashedBananas

# Associativity

`foldl1` and `foldr1`

**What does this equal?**

$$2 \wedge 3 \wedge 4$$

**In what order will this evaluate?**

2 ^ 3 ^ 4

?

$$(2 \wedge 3) \wedge 4 ?$$

*or*

$$2 \wedge (3 \wedge 4) ?$$

# Clue

```
Prelude> :i (^)
(^) :: (Num a, Integral b) => a -> b -> a      -- Defined in `GHC.Real'
infixr 8 ^
```



# Clue

```
Prelude> :i (^)
(^) :: (Num a, Integral b) => a -> b -> a    -- Defined in `GHC.Real'
infixr 8 ^
```



r stands for right associative

# In what order will this evaluate?

$$(2 \wedge 3) \wedge 4 ?$$

*or*

$$2 \wedge (3 \wedge 4) ?$$

# In what order will this evaluate?

$$(8) \quad 4 \text{ ? } = 4096 \quad \text{NO!}$$

*or*

$$2 \text{ ? } (3 \text{ ? } 4) \text{ ?}$$

# In what order will this evaluate?

$$(8) \quad ^4 ? = 4096 \quad \text{NO!}$$

*or*

$$2 \quad ^{(81)} =$$

2417851639229258349412352 YES!

# or according to “numerals”

```
Prelude Text.Numeral.Language.EN Text.Numeral.Grammar.Reified>  
gb_cardinal defaultInflection (2^3^4)
```

```
Just "two septillion four hundred and seventeen sextillion eight hundred  
and fifty-one quintillion six hundred and thirty-nine quadrillion two  
hundred and twenty-nine trillion two hundred and fifty-eight billion three  
hundred and forty-nine million four hundred and twelve thousand three  
hundred and fifty-two"
```

# Common associativities

Left	Right
$+$ $*$ $-$ $/$	<code>`function`</code> $\wedge$ $**$ $:$ $++$ $\&\&$ $  $

# Fold left vs fold right

`foldl1 (^) [2,3,4] == (2 ^ 3) ^ 4`

`foldr1 (^) [2,3,4] == 2 ^ (3 ^ 4)`

# Fold left vs fold right *Continued...*

`foldl1 (-) [2,3,4,5,6] == ((2-3)-4)-5)-6`

`foldr1 (-) [2,3,4,5,6] == (2-(3-(4-(5-6))))))`



Fold **left**

Smallest most inner(Highest precedence) sub-expression on the **left**.

Largest top level sub expression on the **left**.

Fold right

Smallest most inner(Highest precedence) sub-expression on the right.

Largest top level sub expression on the right.

# Fold left vs fold right *Continued...*

return type

$$\text{foldl1 } (-) [2,3,4,5,6] == ((2-3)-4)-5)-6$$

element type

highest precedence operation

return type

element type

The diagram illustrates the foldl1 operation. A red horizontal line is drawn above the expression 'foldl1 (-) [2,3,4,5,6]'. A red arrow points from the label 'return type' to this line. A blue arrow points from the label 'element type' to the list '[2,3,4,5,6]'. To the right of the equals sign, the expression '((2-3)-4)-5)-6' is shown. A red horizontal line is drawn under the sub-expression '(2-3)'. A red arrow points from the label 'highest precedence operation' to this line. Another red arrow points from the label 'return type' to the line under '(2-3)'. A blue arrow points from the label 'element type' to the list '[2,3,4,5,6]'.

return type

$$\text{foldr1 } (-) [2,3,4,5,6] == (2-(3-(4-(5-6))))$$

highest precedence operation

element type

return type

element type

The diagram illustrates the foldr1 operation. A blue horizontal line is drawn above the expression 'foldr1 (-) [2,3,4,5,6]'. A blue arrow points from the label 'return type' to this line. A red arrow points from the label 'element type' to the list '[2,3,4,5,6]'. To the right of the equals sign, the expression '(2-(3-(4-(5-6))))' is shown. A blue horizontal line is drawn under the sub-expression '(5-6)'. A blue arrow points from the label 'highest precedence operation' to this line. Another blue arrow points from the label 'return type' to the line under '(5-6)'. A red arrow points from the label 'element type' to the list '[2,3,4,5,6]'.

# Misnomers

`foldr f v xs /= foldl f v (reverse xs)`

`foldr f v xs /= foldl (flip f) v xs`

# Reasons for choice

Fold right - faster when pattern matching part of result. In the case of infinite list necessary.

Fold left - faster when strictly evaluated.

# How to remember

`foldx :: ? -> ? -> ? -> Result`

- Function
- Initial accumulator
- Structure to fold

In general arguments that are subjectively least likely to be parameterised usually go first to make it easier to use partial application.

# Heterogeneous Folding

$\text{foldl} :: (\textcolor{red}{b} \rightarrow a \rightarrow \textcolor{red}{b}) \rightarrow \textcolor{red}{b} \rightarrow [a] \rightarrow \textcolor{red}{b}$



return type is left argument

$\text{foldr} :: (a \rightarrow \textcolor{blue}{b} \rightarrow \textcolor{blue}{b}) \rightarrow \textcolor{blue}{b} \rightarrow [a] \rightarrow \textcolor{blue}{b}$



return type is right argument