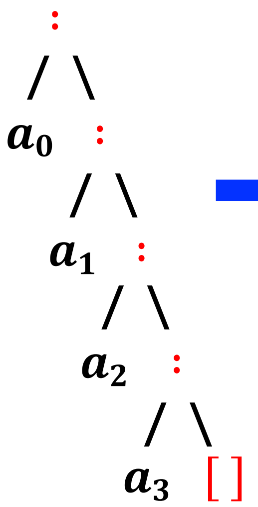
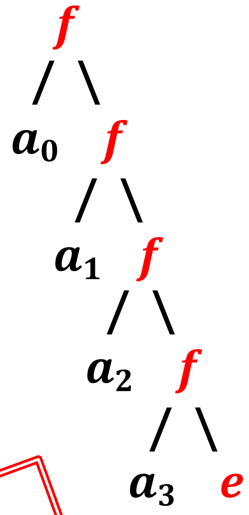


Folding



CHEAT-SHEET

#3

slides by  @philip_schwarz FP Iλλuminated https://fpilluminated.com/

The universal property of $fold$

...

For finite lists, the universal property of $fold$ can be stated as the following equivalence between two definitions for a function g that processes lists:

$$\begin{aligned} g\ [\,] &= v \\ g\ (x : xs) &= f\ x\ (g\ xs) \end{aligned} \quad \Longleftrightarrow \quad g = fold\ f\ v$$

In the right-to-left direction, substituting $g = fold\ f\ v$ into the two equations for g gives the recursive definition for $fold$.

Conversely, in the left-to-right direction the two equations for g are precisely the assumptions required to show that $g = fold\ f\ v$ using a simple proof by induction on finite lists…

Taken as a whole, the universal property states that for finite lists the function $fold\ f\ v$ is not just a solution to its defining equations, but in fact the unique solution….

The universal property of $fold$ can be generalised to handle partial and infinite lists…

A tutorial on the universality and expressiveness of fold

GRAHAM HUTTON

$$\begin{aligned} &fold :: (\alpha \to \beta \to \beta) \to \beta \to ([\alpha] \to \beta) \\ &fold\ f\ v\ [\,] = v \\ &fold\ f\ v\ (x : xs) = f\ x\ (fold\ f\ v\ xs) \end{aligned}$$

Graham Hutton

@haskelhutt

$$\begin{aligned} g\,[\,] &= v \\ g\,(x : xs) &= f\,x\,(g\,xs) \end{aligned} \quad\Longleftrightarrow\quad g = \mathit{fold}\,f\,v$$

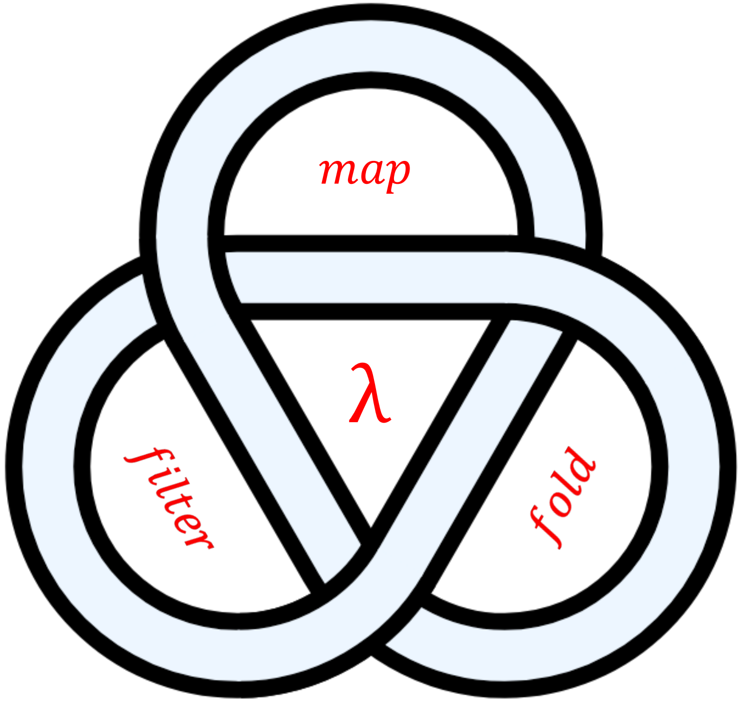
$$\begin{aligned} \mathit{sum} &:: [\mathit{Int}] \to \mathit{Int} \\ \mathit{sum}\,[\,] &= 0 \\ \mathit{sum}\,(x : xs) &= x + \mathit{sum}\,xs \end{aligned} \quad\Longleftrightarrow\quad \mathit{sum} = \mathit{fold}\,(+)\,0$$

$$\begin{aligned} \mathit{product} &:: [\mathit{Int}] \to \mathit{Int} \\ \mathit{product}\,[\,] &= 1 \\ \mathit{product}\,(x : xs) &= x \times \mathit{product}\,xs \end{aligned} \quad\Longleftrightarrow\quad \mathit{product} = \mathit{fold}\,(\times)\,1$$

$$\begin{aligned} \mathit{length} &:: [\alpha] \to \mathit{Int} \\ \mathit{length}\,[\,] &= 0 \\ \mathit{length}\,(x : xs) &= 1 + \mathit{length}\,xs \end{aligned} \quad\Longleftrightarrow\quad \mathit{length} = \mathit{fold}\,(\lambda x.\lambda n.1 + n)\,0$$

$$\begin{aligned} (+\!\!+) &:: [\alpha] \to [\alpha] \to [\alpha] \\ [\,] +\!\!+\, ys &= ys \\ (x : xs) +\!\!+\, ys &= x : (xs +\!\!+\, ys) \end{aligned} \quad\Longleftrightarrow\quad (+\!\!+\, ys) = \mathit{fold}\,(:)\,ys$$

$$\begin{aligned} \mathit{concat} &:: [[\alpha]] \to [\alpha] \\ \mathit{concat}\,[\,] &= [\,] \\ \mathit{concat}\,(xs : xss) &= xs +\!\!+\, \mathit{concat}\,xss \end{aligned} \quad\Longleftrightarrow\quad \mathit{concat} = \mathit{fold}\,(+\!\!+)\,[\,]$$



=

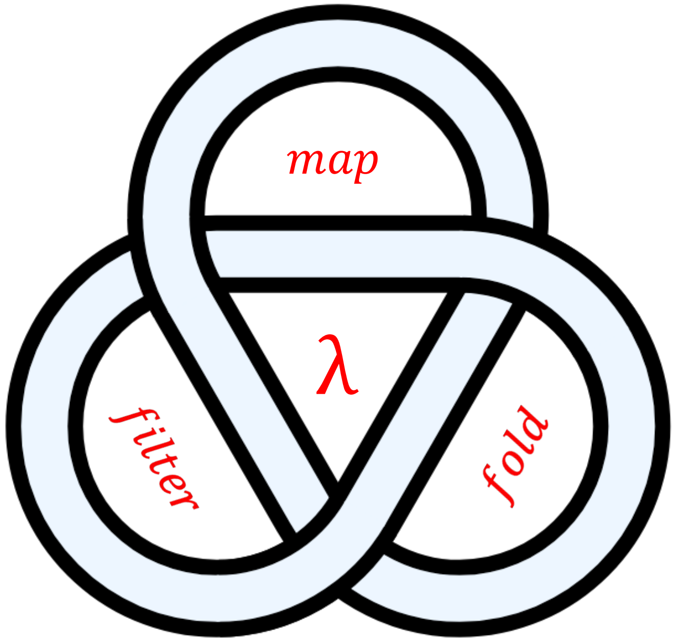
The **Triad** of
map, *filter* and *fold*

The *bread*, *butter*, and *jam* of
Functional Programming

$$\begin{aligned} g\,[\,] &= v \\ g\,(x : xs) &= f\;x\;(g\;xs) \end{aligned} \quad \Longleftrightarrow \quad g = \mathit{foldr}\;f\;v$$

$$\begin{aligned} \mathit{map} &:: (\alpha \to \beta) \to ([\alpha] \to [\beta]) \\ \mathit{map}\;f\;[\,] &= [\,] \\ \mathit{map}\;f\;(x : xs) &= f\;x : \mathit{map}\;f\;xs \end{aligned} \quad \Longleftrightarrow \quad \mathit{map}\;f = \mathit{foldr}\;(\lambda x.\lambda xs.(f\;x) : xs)\;[\,]$$

$$\begin{aligned} \mathit{filter} &:: (\alpha \to \mathit{Bool}) \to ([\alpha] \to [\alpha]) \\ \mathit{filter}\;p\;[\,] &= [\,] \\ \mathit{filter}\;p\;(x : xs) &= \mathbf{if}\;p\;x \\ &\quad\;\mathbf{then}\;x : \mathit{filter}\;p\;xs \\ &\quad\;\mathbf{else}\;\mathit{filter}\;p\;xs \end{aligned} \quad \Longleftrightarrow \quad \mathit{filter}\;p = \mathit{foldr}\;(\lambda x.\lambda xs.\,\mathbf{if}\;p\;x\;\mathbf{then}\;x : xs\;\mathbf{else}\;xs)\;[\,]$$



=

Folding Unfolded

Polyglot FP for Fun and Profit
Haskell and Scala

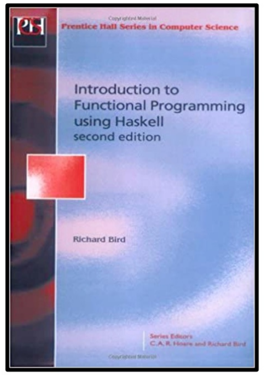
See how **recursive functions** and **structural induction** relate to **recursive datatypes**

Follow along as the **fold abstraction** is introduced and explained

Watch as **folding** is used to simplify the definition of **recursive functions** over **recursive datatypes**

Part 1 - through the work of

inspired by



Richard Bird
http://www.cs.ox.ac.uk/people/richard.bird/

Graham Hutton
@haskellhutt

A tutorial on the universality and expressiveness of fold

GRAHAM HUTTON
University of Nottingham, Nottingham, UK
http://www.cs.nott.ac.uk/~gmh

slides by  @philip_schwarz slideshare https://www.slideshare.net/pjschwarz

FP Illuminated https://fpilluminated.com/