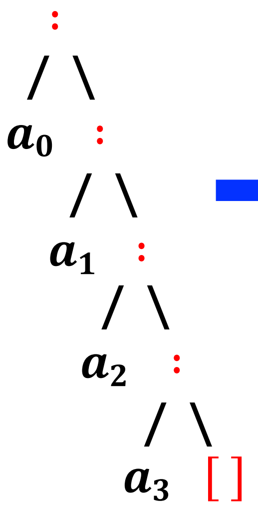
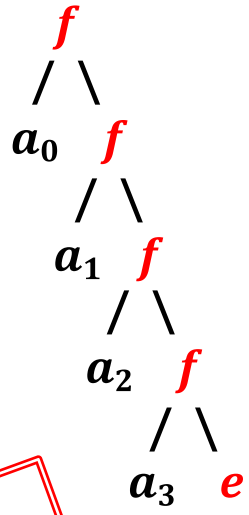
Folding





CHEAT-SHEET #1

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

data $Nat = Zero \mid Succ\ Nat$

Common pattern for many recursive functions over Nat :

$$\begin{aligned} &f :: Nat \to \alpha \\ &f\ Zero = c \\ &f\ (Succ\ n) = h\ (f\ n) \end{aligned}$$

$$\begin{aligned} &c :: \alpha \\ &h :: \alpha \to \alpha \end{aligned}$$

Three examples of such functions:

$$\begin{aligned} &(+) :: Nat \to Nat \to Nat \\ &m + Zero = m \\ &m + (Succ\ n) = Succ\ (m + n) \end{aligned}$$

$$\begin{aligned} &(\times) :: Nat \to Nat \to Nat \\ &m \times Zero = Zero \\ &m \times (Succ\ n) = (m \times n) + m \end{aligned}$$

$$\begin{aligned} &(\uparrow) :: Nat \to Nat \to Nat \\ &m \uparrow Zero = Succ\ Zero \\ &m \uparrow (Succ\ n) = (m \uparrow n) \times m \end{aligned}$$

The common pattern can be captured in a function:

$$\begin{aligned} &foldn :: (\alpha \to \alpha) \to \alpha \to Nat \to \alpha \\ &foldn\ h\ c\ Zero = c \\ &foldn\ h\ c\ (Succ\ n) = h\ (foldn\ h\ c\ n) \end{aligned}$$

The three sample functions implemented using $foldn$:

$$\begin{aligned} &m + n = foldn\ Succ\ m\ n \\ &m \times n = foldn\ (\lambda x.\ x + m)\ Zero\ n \\ &m \uparrow n = foldn\ (\lambda x.\ x \times m)\ (Succ\ Zero)\ n \end{aligned}$$

data $List\ \alpha = Nil \mid Cons\ \alpha\ (List\ \alpha)$

Common pattern for many recursive functions over $List$:

$$\begin{aligned} &f :: List\ \alpha \to \beta \\ &f\ Nil = c \\ &f\ (Cons\ x\ xs) = h\ x\ (f\ xs) \end{aligned}$$

$$\begin{aligned} &c :: \beta \\ &h :: \alpha \to \beta \to \beta \end{aligned}$$

Three examples of such functions:

$$\begin{aligned} &sum :: List\ Nat \to Nat \\ &sum\ Nil = Zero \\ &sum\ (Cons\ x\ xs) = x + (sum\ xs) \end{aligned}$$

$$\begin{aligned} &length :: List\ \alpha \to Nat \\ &length\ Nil = Zero \\ &length\ (Cons\ x\ xs) = Succ\ Zero + (length\ xs) \end{aligned}$$

$$\begin{aligned} &append :: List\ \alpha \to List\ \alpha \to List\ \alpha \\ &append\ Nil\ ys = ys \\ &append\ (Cons\ x\ xs)\ ys = Cons\ x\ (append\ xs\ ys) \end{aligned}$$

The common pattern can be captured in a function:

$$\begin{aligned} &foldr :: (\alpha \to \beta \to \beta) \to \beta \to [\alpha] \to \beta \\ &foldr\ f\ b\ Nil = b \\ &foldr\ f\ b\ (Cons\ x\ xs) = f\ x\ (foldr\ f\ b\ xs) \end{aligned}$$

The three sample functions implemented using $foldr$:

$$\begin{aligned} &sum\ xs = foldr\ (+)\ Zero\ xs \\ &length\ xs = foldr\ (\lambda x.\lambda n.\ n + (Succ\ Zero))\ Zero\ xs \\ &append\ xs\ ys = foldr\ Cons\ ys\ xs \end{aligned}$$

Folding Unfolded

Polyglot FP for Fun and Profit
Haskell and Scala

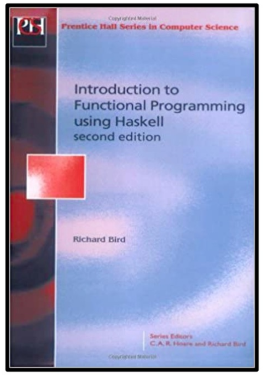
See how **recursive functions** and **structural induction** relate to **recursive datatypes**

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 Iλλuminated https://fpilluminated.com/