A **functor** is an operator that maps types to types

$$X \text{ type} \Rightarrow FX \text{ type}$$

and functions to functions

$$f : A \to B \Rightarrow Ff : FA \to FB$$

It must preserve identities and compositions

$$A \xrightarrow{id_A} A \Rightarrow FA \xrightarrow{Fid_A} FA$$

$$Fid_A = id_{FA}$$

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[rr, bend right, "g \circ f"'] & B \arrow[r, "g"] & C \end{tikzcd} \Rightarrow \begin{tikzcd} FA \arrow[r, "Ff"] \arrow[rr, bend right, "F(g \circ f)"'] & FB \arrow[r, "Fg"] & FC \end{tikzcd}$$

$$F(g \circ f) = Fg \circ Ff$$

Strictly Positive Functor

FX is defined by an expression where X occurs only on the right of arrows

Examples:

$FX = \mathbb{1} + X$

$FX = \mathbb{1} + A \times X$ *for fixed A*

$FX = A + X \times X$

$FX = (X \to A) \to A$

↳ not strictly positive

Every strictly positive functor specifies the structure of a recursive type μF

- $FX = \mathbb{1} + X$

one constant initial element ⇓ 0

one unary constructor ⇓ succ

μF corresponds to Nat

- $FX = \mathbb{1} + A \times X$

one constant ⇓ nil

unary constructor with parameter A ⇓ (::)

μF correspond to $List_A$

- $FX = A + X \times X$

a constant for every element of A ⇓ leaf

binary constructor ⇓ node

μF corresponds to $Tree_A$

Rules for the Inductive Type μF

Introduction

$$\frac{t : F\,\mu F}{\mathrm{in}_F\, t : \mu F}$$

- Example: if $FX = \mathbb{1} + X$ then $\mu F \cong \mathrm{Nat}$

  The rule says:

  if $t : \mathbb{1} + \mathrm{Nat}$

  then $\mathrm{in}\, t : \mathrm{Nat}$

Elements of $F\,\mu F = \mathbb{1} + \mathrm{Nat}$

$$\begin{array}{ccc} \mathbb{1} & + & \mathrm{Nat} \\ \ni & & \ni \\ \mathrm{inl}\, * & & \mathrm{inr}\, n \\ \| & & \| \\ \mathrm{zero} & & \mathrm{succ}\, n \end{array}$$

$*$: only element of $\mathbb{1}$

n: any element of Nat

- Example: if $FX = \mathbb{1} + A \times X$ then $\mu F \cong \mathrm{List}_A$

  Elements of

$$\begin{array}{ccccc} F\,\mu F & = & \mathbb{1} & + & A \times \mathrm{List}_A \\ & & \ni & & \ni \\ & & \mathrm{inl}\, * & & \mathrm{inr}\, \langle a, \ell \rangle \\ & & \| & & \| \\ & & \mathrm{nil} & & a :: \ell \end{array}$$

Elimination

For every type X

$$\frac{f : FX \longrightarrow X}{\mathrm{cata}\ f : \mu F \longrightarrow X}$$

Reduction

$$\mathrm{cata}\ f\ (\mathrm{in}\ t)$$

$$\rightsquigarrow f\,(F\,(\mathrm{cata}\ f)\ t)$$

Explanation:

f tells us how to compute on the constructors and the recursive calls

$\mathrm{cata}\ f : \mu F \to X$ is called the catamorphism of f

Since the functor F can be applied to functions

$$\mathrm{cata}\ f : \mu F \to X \Rightarrow F(\mathrm{cata}\ f) : F\mu F \longrightarrow FX$$

So we can apply it to $t : F\mu F$

$$F\,(\mathrm{cata}\ f)\ t : FX$$

If we now apply f to it:

$$f\,(F\,(\mathrm{cata}\ f)\ t) : X$$

The elimination principle corresponds to iteration: we iterate f down the structure.

- Example:

  For $FX = \mathbb{1} + X$

  the elimination rule says

$$\frac{f : \mathbb{1} + X \longrightarrow X}{\mathrm{cata}\, f : \mathrm{Nat} \longrightarrow X}$$

equivalent to

$$\frac{x_0 : X \qquad g : X \longrightarrow X}{\mathrm{iterate}\; x_0\; g : \mathrm{Nat} \to X}$$

$$\mathrm{iterate}\; x_0\; g\; n \leadsto^{*} \underbrace{g(g \cdots (g}_{n \text{ times}} x_0))$$

$x_0 = f(\mathrm{inl}\, *)$

$g = \lambda x.\, f(\mathrm{inr}\, x)$

- Example

  For $FX = \mathbb{1} + A \times X$

  the elimination rule says

$$\frac{f : \mathbb{1} + A \times X \longrightarrow X}{\mathrm{cata}\, f : \mathrm{List}_A \longrightarrow X}$$

equivalent to

$$\frac{x_0 : X \qquad g : A \to X \to X}{\mathrm{iterate}\; x_0\; g : \mathrm{List}_A \longrightarrow X}$$

(called foldr in Haskell)