



Fold

$Nat := zero \mid succ(Nat)$

Nat is the set of Natural numbers (zero, succ(zero), succ(succ(zero)), ...). This set has the following structure:

- Order- each number is less or more than any other member. For any $m, n \in Nat$:
Ord(m, n) is
 - $m > n$
 - or
 - $m < n$
 - or
 - $m = n$

A homomorphism $f: Nat \rightarrow Nat$ is a structure preserving map: Ord(m, n) = Ord($f(m), f(n)$). Two homomorphisms under function composition forms a homomorphism too.

Following is a recursive definition of a function $f: Nat \rightarrow A$ (A is some type)

- $f(0) = c$
 $f(n+1) = h(f(n))$, where $c : A$ and $h: A \rightarrow A$

The above definition is known as **Structural Recursion**.

Some examples:

- Plus : $Nat \times Nat \rightarrow Nat$
Plus($m, 0$) = m
Plus($m, n+1$) = succ(Plus(m, n)), where succ(n) = $n+1$

Some properties of Plus

- Plus(m, n) = m or Plus(m, n) = n
- or
- Plus(m, n) > m and Plus(m, n) > n
- Plus(m, n) = Plus(n, m)

Plus can be also be considered in curried form- Plus_m : $Nat \rightarrow Nat$

Plus_m (0) = m

Plus_m ($n+1$) = succ(Plus_m(n)), where succ(n) = $n+1$

Ord(o, n) = Ord(Plus_m(o), Plus_m(n)), where $o, n \in Nat$. Plus_m preserves order and it is a homomorphism.

- Multiply : $Nat \times Nat \rightarrow Nat$
Multiply($m, 0$) = 0
Multiply($m, n+1$) = Plus($m, Multiply(m, n)$)

Some properties of Multiply

- Multiply(m, n) = m or Multiply(m, n) = n
- or
- Multiply(m, n) > m and Multiply(m, n) > n
- Multiply(m, n) = Multiply(n, m)

Multiply can be also be considered in curried form- $\text{Multiply_m} : \text{Nat} \rightarrow \text{Nat}$

$\text{Multiply_m}(0) = 0$

$\text{Multiply_m}(n+1) = \text{Plus_m}(\text{Multiply_m}(n))$

$\text{Ord}(o,n) = \text{Ord}(\text{Multiply_m}(o), \text{Multiply_m}(n))$, where $o,n \in \text{Nat}$. Multiply_m preserves the order and it is a homomorphism.

- $\text{Square} : \text{Nat} \rightarrow \text{Nat}$

$\text{Square}(0) = 0$

$\text{Square}(n+1) = h(\text{Square}(n))$, where $h(\text{Square}(n)) = \text{Square}(n) + 2 * n + 1 = \text{succ}(\text{Plus}(\text{Multiply_2}(n), \text{Square}(n)))$

$\text{Ord}(o,n) = \text{Ord}(\text{Square}(o), \text{Square}(n))$, where $o,n \in \text{Nat}$. Square preserves the order and it is a homomorphism.

From these examples an the repeating behavior can be encapsulated as: $\text{foldn} : A \rightarrow (A \rightarrow A) \rightarrow (\text{Nat} \rightarrow A)$

- $\text{foldn}(c, h)$ creates a function that takes a natural number (expressed using *zero* and *succ*) and substitutes *c* for *zero* and *h* for *succ* before evaluating.

$\text{List } A ::= \text{nil} \mid \text{cons}(A, \text{List } A)$

This type has similar recursive structure like Nat :

| <i>List A</i> | <i>Nat</i> |
|--|-----------------------|
| nil | zero |
| cons(a, nil) | succ(zero) |
| cons(a ₁ , cons(a,nil)) | suc(succ(zero)) |
| ... | ... |
| cons(a _n , ...(cons(a,nil)) | succ(...(succ(zero))) |

It should be possible to have a function *f* that is defined recursively over this datatype

f: $\text{List } A \rightarrow B$

- $f(\text{nil}) = c$
- $f(\text{cons}(a,x)) = h(a, f(x))$, where $c: B, h: A \times B \rightarrow B$

$\text{foldr} : B \rightarrow (A \times B \rightarrow B) \rightarrow (\text{List } A \rightarrow B)$ is an encapsulation of this recursion over this datatype (just like foldn over Nat).

- $\text{foldr}(c,h)$ creates a function that takes List A (expressed using *nil* and *cons*) and substitutes *c* for *nil* and *h* for *cons* before evaluating

Follows example functions that can be define recursively over List and its equivalent using foldr

- $\text{map} : \text{List } A \rightarrow (A \rightarrow B) \rightarrow \text{List } B$
 - $\text{map } f \text{ nil} = \text{nil}$
 - $\text{map } f \text{ cons}(a, x) = \text{cons}(f(a), \text{map } f x)$
 - In terms of foldr
 - $\text{map } f = \text{foldr}(\text{nil}, h)$, where $h(a,x) = \text{cons}(f(a), x)$

- $\text{length} : \text{List } A \rightarrow \text{Nat}$
 - $\text{length nil} = 0$
 - $\text{length cons}(a, x) = \text{length}(x) + 1$
 - In terms of foldr
 $\text{length} = \text{foldr}(0, h)$, where $h(a, n) = n + 1$
- $\text{sum} : \text{List } A \rightarrow \text{Nat}$
 - $\text{sum nil} = 0$
 - $\text{sum cons}(a, x) = a + \text{sum}(x)$
 - In terms of foldr
 $\text{sum} = \text{foldr}(0, \text{Plus})$, where $\text{Plus}(a, b) = a + b$