

Categories and Reason

Functors

Mutaamba Maasha

F# Users Group

Ever Ponder?

- fold f [] list : Why does fold look like this?(Example)
- What is this category theory business?

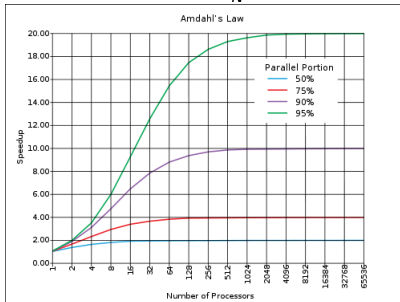
- Software Won
 - Formal Methods : The Use of Industrial-Strength Formal Methods: Jonathan P. Bowen , Michael G. Hinchey : 1997
 - Physical becoming software
- Transistors are becoming cheaper and systems are becoming more complex
 - Medicine
 - MRI - Robotic Surgery
 - Transportation
 - Boeing 777 - Hybrid vehicles - High Speed Trains

Amdahl's Law

P = percentage of program that can be parallelized

N = number of processors

$$\text{speedup} = \frac{1}{(1-p) + (\frac{p}{N})}$$



Function Curry/Schönfinkel

Transform $f : (X \times Y) \rightarrow Z$ to $f : X \rightarrow Y \rightarrow Z$
 $((X \times Y) \rightarrow Z) \rightarrow f : X \rightarrow Y \rightarrow Z$

```
let curry f = fun x y -> f (x,y);;  
let uncurry f = fun (x,y) -> f x y;;
```

We'll meet Mister Haskell Curry and F# later.

- INRA
- proof assistant
- Gallina

- Objects : Types
- Arrows : Functions
 - Morphisms $A \rightarrow B$
 - Domain $A \rightarrow$ Codomain B
- Composition
 - Associativity : $f : A \rightarrow B$ and $g : B \rightarrow C$ and $h : C \rightarrow D$:
 $f \circ (g \circ h) = (f \circ g) \circ h$
 - Identity: : $f : A \rightarrow B$ there is id_A and id_B where
 $id_B \circ f = f \circ id_A = f$

(Co) Products

- Product

- $A \leftarrow A \times B \rightarrow B$
- $A \times B = \{(a, b) | a \in A \wedge b \in B\}$

- Coproduct

- $A \rightarrow A + B \leftarrow B$
- $A + B = \{(0, a) | a \in A\} \cup \{(1, b) | b \in B\}$

- Identity

- $A + 0 \cong 0 + A \cong A$
- $A \times 1 \cong 1 \times A \cong A$
- $0 = \emptyset$
- $1 = \{()\} = \{void\}$

Natural Numbers

```
Module Nat. Inductive nat : Set :=  
  | 0 : nat  
  | S : nat -> nat.  
End Nat.
```

Sample

```
Eval simpl in (S 0).  
Eval simpl in (S (S 0)).
```

```
Inductive list (A:Set) : Set :=  
  | nil : list A  
  | cons : A -> list A -> list A.
```

Sample

```
Eval simpl in nil.
```

```
Eval simpl in (cons nat 2 (cons nat 1 (nil nat))).
```

```
Inductive tree (A:Set) : Set :=  
  | leaf    : tree A  
  | branch  : A -> tree A -> tree A -> tree A.
```

Sample

```
Eval simpl in leaf nat.
```

```
Eval simpl in branch nat 1 (leaf nat) (leaf nat).
```

Natural Numbers

`nat zero : () = 0`

`zero : 1 \rightarrow nat`

`nat S : n = S(n)`

`S : nat \rightarrow nat`

The functions [zero,S] form an algebra of the functor $F(X) = 1 + X$. *nat* is called the carrier of the algebra. F is called the structure of the algebra. ($[zero,S], 1 + X$)

A and B be categories. A functor is : $F : A \rightarrow B$

1. $F(id_A) = id_B$

2. for all morphisms $f : A \rightarrow B$ and

$g : B \rightarrow C : F(g \circ f) = F(g) \circ F(f)$

$\text{nil} : () = \text{nil}$

$\text{nil} : 1 \rightarrow \text{List}$

$\text{cons} : (a, l) = \text{cons}(a, l)$

$\text{cons} : A \times \text{List} \rightarrow \text{List}$

$F : [\text{nil}, \text{cons}] : 1 + A \times X$

A Tree leaf : () = leaf
leaf : 1 \rightarrow Tree A
branch : (a,l,r) = branch(a,l,r)
branch : A \times Tree \times Tree \rightarrow Tree
F : [leaf,branch] : 1 + A \times X \times X

Morphisms

Morphisms : Map between two objects in category theory

Initial Algebra : $A^* = \bigcup_{n \in \mathbb{Z}} F^n(A)$

Nat Catamorphism

$nat : ([zero, S], 1 + X)$

$B : ([1, \lambda x. 2 * x], 1 + X)$

$B^* = \{2^n : n \in nat\}$

$$\begin{array}{ccc} 1 + N & \xrightarrow{id+h} & 1 + B^* \\ \downarrow [zero, S] & & \downarrow [1, \lambda x. 2 * x] \\ N & \xrightarrow{h} & B^* \end{array}$$

Fixpoint foldNat

(A B : Set) (f: B -> B)

(b: B) (n : nat) {struct n} : B :=

match n with

| 0 => b

| S n' => f (foldNat A B f b n')

end.

Definition powerOfTwo (n : nat) :=

foldNat nat nat doubleNat 1 n.

Definition doubleNat (n: nat) : nat :=

plus n n.

Unit Testing

Example `test_powerOfTwo_0`: `powerOfTwo 0 = 1`.

Proof.

`simpl.`

`reflexivity.`

Qed.

Example `test_powerOfTwo_3`: `powerOfTwo 3 = 8`.

Proof.

`compute.`

`reflexivity.`

Qed.

List Catamorphism

$([nil, cons]: 1 + A \times X)$
 $[0, \lambda x. 1 + \pi_1 x]$
 $B^* = \{2^n : n \in \text{nat}\}$

$$\begin{array}{ccc} 1 + A \times X & \xrightarrow{id + id \times h} & 1 + A \times N \\ \downarrow [nil, cons] & & \downarrow [0, \lambda x. 1 + \pi_1 x] \\ A^* & \xrightarrow{h} & N \end{array}$$

```
Fixpoint foldList (A B :Set) (f: A -> B -> B) (b: B)
  (l:list A) {struct l} : B :=
match l with
| nil      => b
| cons h t => f h (foldList A B f b t)
end.
```

```
Definition lenghtListFunction (value : nat)
  (acc : nat) : nat := S(acc).
```

```
Definition lengthList (l : list nat) : nat :=
foldList nat nat lenghtListFunction 0 l.
```

Unit Proving

```
Theorem length_increase :  
  forall (l:list nat) (n: nat)  
    , lengthList (cons nat n l) = S(lengthList l).
```

Proof.

```
  intros l n.  
  induction l as [| l'].  
  Case "l = nil".  
    unfold lengthList.  
    simpl.  
    unfold lenghtListFunction.  
    reflexivity.  
  Case "l = cons".  
    unfold lengthList.  
    simpl.  
    unfold lenghtListFunction.  
    reflexivity.
```

Qed.

Map Polemic

```
Definition mapList (A B : Set) (f: A -> B) (l : list A) :=
  foldList A (list B)
    (fun (value : A)
      (acc : list B) => (cons B (f value) acc))
    (nil B).
```

Check mapList.

```
mapList
  : forall A B : Set,
  (A -> B) ->
  list A -> list A -> list B
```

Reduce

Domain = CoDomain

Show : $f(x, f(y, z)) = f(f(x), f(y, z))$

$map\ f \circ map\ g = map\ f \circ g$

$fold\ f\ (map\ g) = fold\ f.g$

Tree Catamorphism

$$\begin{array}{ccc} 1 + N \times A^* \times A^* & \xrightarrow{id+h+h} & 1 + N \times N \times N \\ \downarrow [leaf, branch] & & \downarrow [0, \lambda x. x_1 * (x_2 + x_3)] \\ N & \xrightarrow{h} & N \end{array}$$

```
Fixpoint foldTree (A B :Set) (l:tree A)
  (f: (A * B * B) -> B) (b: B) {struct l} : B :=
match l with
| leaf          => b
| branch a l r => f(a,(foldTree A B l f b),(foldTree A B l f b))
end.
```

Others

1. Anamorphism : $X \rightarrow 1 + A \times X$
2. Hylomorphism : composition of an anamorphism and catamorphism.

For Further Reading I



Bart Jacobs , Jan Rutten.

Tutorial on (Co)Algebras and (Co)Induction

EATCS Bulletin, 62:62–222, 1997.



Wikipedia contributors,

Amdahl's law

Wikipedia, The Free Encyclopedia,

http://en.wikipedia.org/w/index.php?title=Amdahl%27s__law&oldid=

(accessed January 3, 2010).



Safety critical systems: challenges and directions Knight, J.C. ,

Software Engineering, 2002. ICSE 2002.