Computing with Categories

Saul Youssef
Center for Computational Science
Boston University
April 2001

Sets

Preorders

Graphs

Finite State Machines

Monoids

Groups

Vector Spaces

Metric Spaces

Measure Space

Topological Space

Manifolds

Lie Groups

Fiber Bundles

⋮

Amazingly, these all have a rich common structure. They are all **Categories**.

A category consists of

(a) A collection of **objects**.

(b) For each pair A, B of objects, a set **Hom(A,B)** of **morphisms**.

(c) An associative rule for composing morphisms

$$\circ : \mathrm{Hom}(A,B) \times \mathrm{Hom}(B,C) \to \mathrm{Hom}(A,C)$$

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[rr, bend right, "g \circ f"'] & B \arrow[r, "g"] & C \end{tikzcd}$$

(d) An identity morphism $A \xrightarrow{i_A} A$ for each object, such that

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[loop below, "i_A"] & B \arrow[loop below, "i_B"] \end{tikzcd}$$ "commutes" for any f.

..............................functions

..............................monotonic functions

..............................graph morphisms

.....automata morphisms

..............................monoid morphisms

..............................group homomorphisms

...............linear maps

..............contraction mappings

...............measure morphisms

..........continuous functions

...................smooth functions

...................smooth group homomorphs

............bundle morphisms

⋮

$$A \xleftarrow{\alpha} A \times B \xrightarrow{\beta} B$$

cartesian product
group direct product
vector space product
⋮

$$A \xrightarrow{\alpha} A \oplus B \xleftarrow{\beta} B$$

disjoint union
group direct sum
vector space product
⋮

$$A \xrightarrow{\text{monic}} X$$

Subset
subgroup
subspace of a vector space
⋮

$$A \xleftarrow{\text{epic}} X$$

equivalence classes
quotient groups
quotient topologies
⋮

$1 \xrightarrow{x} X$ "element of X"

An element of a set is a member of the set.
An element of a graph is a loop in the graph.
An element of an automaton is a fixed point.
An element of a fiber bundle is a section.

$$\begin{tikzcd} X \arrow[r, "s"] \arrow[dr] & E \arrow[d, "\pi"] \\ & X \end{tikzcd}$$

A **Functor** from category C to category C' consists of

(a) A map ($\mathcal{F}$) from the objects of C to the objects of C'.

(b) For each pair of objects A, B, a map

$$\mathcal{F} : \mathrm{Hom}_C(A, B) \to \mathrm{Hom}_{C'}(\mathcal{F}A, \mathcal{F}B)$$

preserving identities and composition

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[rr, bend right, "g \circ f"'] & B \arrow[r, "g"] & C \end{tikzcd} \qquad \begin{tikzcd} \mathcal{F}A \arrow[r, "\mathcal{F}f"] \arrow[rr, bend right, "\mathcal{F}(g \circ f)"'] & \mathcal{F}B \arrow[r, "\mathcal{F}g"] & \mathcal{F}C \end{tikzcd}$$

<u>examples:</u>

$\mathcal{P}$ Power set

V^* Dual of a vector space

$T_x M$ Tangent space of a manifold

π_x Homotopy

Second Quantization

Λ^* cohomology functor

Functors often come in **Adjoint** pairs

$$\mathcal{L} : C \to C'$$

$$\mathcal{R} : C' \to C$$

$$\mathrm{Hom}_{C'}(\mathcal{L}A, B) \cong \mathrm{Hom}_{C}(A, \mathcal{R}B)$$

The basis of a vector space.

The free group on a set

$G \mapsto G/[G,G]$ commutator subgroup

Universal enveloping algebra of a Lie Algebra

Completion of a metric space.

D.E. Rydeheard + R.M. Burstall

Computational Category Theory, Prentice-Hall 1988

Tatsuya Hagino, Ph.D. Edinburgh University 1987

R.F.C. Walters, Categories and Computer Science

Cambridge 1991

"Magma" Computational algebra system, U. Sydney.

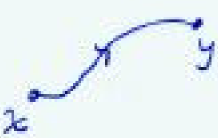
"Charity" Robin Cockett et al. U. of Calgary

Scientific & Engineering C++ Barton & Nackman

1994

Why can't we program this way?

What type should represent a category?
It should include the following as special cases:

Objects	Morphisms
Sets	functions
integers	$n \times m$ matrices
points in a topological space	 smooth paths in the space modulo homotopy
elements of a preorder	$p \leq q$
categories	functors
functors	natural transformations
$A \xrightarrow{f} B$	see the diagram below

Morphisms for the objects $A \xrightarrow{f} B$:

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "\alpha"'] & B \arrow[d, "\beta"] \\ A' \arrow[r, "f'"] & B' \end{tikzcd}$$

where the diagram commutes

+ many more

Aldor

Offshoot of AXIOM by Stephen Watt and collaborators. Now under development at N.A.G.

Domains ($\cong$ Classes in OO languages) are first class objects - may be created by functions, passed as parameters etc.

Domains have types called Categories which are also first class.

```
Group : Category == with
    1 : %
    * : (%, %) -> %
    inv : % -> %
```

n.b. an Aldor category is not the same as a mathematical category

Parametric Polymorphism + Dependent Types + Curried functions

Domain

Category

```
commutator (R: Ring)(x: R, y: R) : R
        == x*y - y*x
```

The syntax is made to resemble mathematics:

```
A: AbelianGroup == Integer add
f: A -> A == (a: A): A +-> a + a
```

The type system is uniform:

```
i: Integer == 5
extend Integer: with Group == add
```

even

```
->: TupleType -> TupleType
```

!

Compiles to C, machine independent "FOAM" or Lisp. Heavily optimized.

Many Styles are Possible

Objects / Morphs.	Type	Category	%
$\longrightarrow$			
Hom(A,B)			
%			

+ Choices for

    Functor

    Natural transformation

    Adjoint

One style as an example:

Objects in a particular category are the collection of domains satisfying some Obj: Category.

Morphisms from A: Obj to B: Obj are objects satisfying A -> B. Properties of the morphisms are implicit as with the associativity of * in group.

```
MathCategory(Obj: Category): Category == with
    id: (A: Obj) -> (A -> A)
    compose: (A: Obj, B: Obj, C: Obj) ->
        (A -> B, B -> C) -> (A -> C)
    default
      id(A: Obj): (A -> A) == (a: A): A +-> a
      compose(A: Obj, B: Obj, C: Obj)(f: A -> B, g: B -> C): (A -> C)
        == (a: A): C +-> g(f(a))
```

Functors from $\mathrm{Obj}\, A$ to $\mathrm{Obj}\, B$ are domain constructors like

$$\mathrm{Left} : \mathrm{Obj}\, A \to \mathrm{Obj}\, B$$

where the action of Left on morphisms is fixed by a suitable domain satisfying

```
RightAdjoint (Obj A : Category,
              Obj B : Category,
              Left : Obj A -> Obj B)

    <<: } Hom (Left A, B) ≅ Hom (A, Right B)
    >>:

    default

      ~~left~~2:
      left : (A : Obj A, B : Obj A, f : A -> B)
                   -> (Left A -> Left B)

            == ....
```

Composition, Identities, Functor actions on morphisms are all supplied by default.

```
#include "base.as"

define Grp:Category == Monoid with
    inv: % -> %
```

← Group definition

```
Forget(G:Grp):Set == G add
```

← Forgetful functor

```
FreeGroup(S:Set):Grp == add
```

← Free Group

```
FreeGroup:FreeConstruction(Set,Grp,Forget) == add


GroupCategory: MathCategory Grp with _
                    Product Grp with _
                  CoProduct Grp with _
                      Final Grp with _
                    Initial Grp
== add
```

The category of Groups

```
"grp.as", line 11: FreeGroup:FreeConstruction(Set,Grp,Forget) == add
                   ............................................^
[L11 C47] #1 (Error) The domain is missing some exports.
        Missing >>: (A: Set, B: Grp, Left(A) -> B) -> A -> Forget(B)
        Missing <<: (A: Set, B: Grp, A -> Forget(B)) -> Left(A) -> B
        Missing Left: Set -> Grp

"groupdemo.as", line 18: == add
                   ...^
[L18 C4] #2 (Error) The domain is missing some exports.
        Missing Product: (A: Grp, B: Grp) -> (AB: Grp, AB -> A, AB -> B, (X: Grp
) -> (X -> A, X -> B) -> X -> AB)
        Missing CoProduct: (A: Grp, B: Grp) -> (AB: Grp, A -> AB, B -> AB, (X: G
rp) -> (A -> X, B -> X) -> AB -> X)
        Missing 1: Grp
        Missing 1: (A: Grp) -> A -> 1
        Missing 0: Grp
        Missing 0: (A: Grp) -> 0 -> A
```

```
#include "base.as"

#library lMonoids          "Monoids.ao"
#library lSets             "Sets.ao"

import from lMonoids,lSets

define Grp:Category == Monoids with
    inv: % -> %

Forget(G:Grp):Set == G add

FreeGroup(A:Set):Grp == add
    Rep == List Record(dom:A,inv:Boolean); import from Rep,SingleInteger
    (t:TextWriter)<<(x:%):TextWriter ==
        (w:TextWriter)**(r:Record(dom:A,inv:Boolean)):TextWriter ==
            r.inv => t << r.dom << "'"
            t << r.dom
        l:List Record(dom:A,inv:Boolean) == rep x
        empty? l => t
        for xx:Record(dom:A,inv:Boolean) in l repeat
            t := t ** xx
        t
    1:% == per [ ]
    inv(x:%):% ==  per [[((rep x).i).dom,~(((rep x).i).inv)]
                       for i:SingleInteger in #(rep x)..1 by -1]
    (x:%)=(y:%):Boolean ==
        #(rep x)=#(rep y) => forall? ( (rep x).i = (rep y).i _
                       for i:SingleInteger in 1..#(rep x))
        false
    (x:%)*(y:%):% ==
        IRep == Record(dom:A,inv:Boolean); import from IRep
        xl:List IRep == rep x
        yl:List IRep == rep y
        cancel(a:IRep,b:IRep):Boolean == a.dom = b.dom and a.inv ~= b.inv
        mult(l:List IRep,r:List IRep):List IRep ==
            empty? l => r
            empty? r => l; S ==> SingleInteger; import from S
            if   cancel(last l,first r) then
                 { mult([l.i for i:S in 1..(#l)-1],[r.i for i:S in 2..#r]) }
            else { concat(l,r) }
        per mult(xl,yl)

+++
+++  Free construction adjoint
+++
FreeGroup:FreeConstruction(Set,Grp,Forget) == add
    Left(A:Set):Grp == FreeGroup(A) add
    <<(A:Set,B:Grp,f:A->Forget B):(Left A->B) ==
        F(a:Left A):B ==
            RepG == List Record(dom:A,inv:Boolean); import from RepG
            l:List B == [(f (x.dom)) pretend B for x in (a pretend RepG)]
            monoidProduct l
        F
    >>(A:Set,B:Grp,f:Left A -> B):(A->Forget B) ==
        F(a:A):Forget B ==
            RepG == List Record(dom:A,inv:Boolean); import from RepG
            aa:Left A == [a,true] pretend Left A
            (f aa) pretend Forget B
```

F

```
+++
+++  The Category of Grps
+++
GroupCategory:MathCategory Grp _
    with            Initial Grp _
    with              Final Grp _
    with            Product Grp _
    with          CoProduct Grp
== add
---
---    In the category of Grp, the trivial group {1} is both initial and final.
---
    1:Grp == (1$MonoidCategory) add { inv(x:%):% == x }
    1(A:Grp):(A->1) == (a:A):1 +-> 1$1
    0:Grp == 1 add
    0(A:Grp):(0->A) == (z:0):A +-> 1$A
---
---    The Group Direct Product
---
    Product(A:Grp,B:Grp):(AB:Grp,AB->A,AB->B,(X:Grp)->(X->A,X->B)->(X->AB)) ==
        (P:Monoids,pa:P->A,pb:P->B,product:(X:Monoids)->(X->A,X->B)->(X->P)) ==
                 (Product$MonoidCategory)(A,B)
        extend P: with Grp == P add
            Rep == Record(a:A,b:B); import from Rep
            inv(x:%):% == per [ inv ((rep x).a), inv ( (rep x).b ) ]
        (P,pa,pb,product)
---
---    The sum of Grp A and B is the free group on the
---    disjoint union of G and H as Sets.
---
    CoProduct(A:Grp,B:Grp):(AB:Grp,A->AB,B->AB,(X:Grp)->(A->X,B->X)->(AB->X))==
        (S:Set,ia:A->S,ib:B->S,coproduct) == (CoProduct$SetCategory)(A,B)
        import from FreeGroup
        SG:Grp == FreeGroup A
        u:S->SG == unit(S) pretend (S->SG)
        coproduct2(X:Grp)(f:A->X,g:B->X):SG->X ==
            ff:S->Forget X == coproduct(X)(f,g) pretend (S->Forget X)
            (<<(S,X,ff)) pretend (SG->X)
        import from o(Set,A,S,SG),o(Set,B,S,SG)
        (SG,u**ia,u**ib,coproduct2)
```

$$
\begin{tikzcd}
G \arrow[r] \arrow[d, leftrightarrow] \arrow[ddr, bend right, "f"'] & G \cup\!\!\cdot\, H \arrow[d] & H \arrow[l] \arrow[d, leftrightarrow] \\
G \arrow[r] & \mathrm{Free} \arrow[d] & H \arrow[l] \arrow[dl, bend left, "g"] \\
& X &
\end{tikzcd}
$$

```
Freyd(ObjA:Category,CatA:MathCategory ObjA,
      ObjB:Category, Left:ObjA->ObjB,
      Ad:RigthAdjoint(ObjA,ObjB,Left)):
                                        MathCategory ObjB  with
       if CatA has     Initial ObjA then     Initial ObjB
       if CatA has   CoProduct ObjA then   CoProduct ObjB
       if CatA has CoEqualizer ObjA then CoEqualizer ObjB
       if CatA has Pushforward ObjA then Pushforward ObjB
== add
      ...
```

Adjoint functor theorem:

Right adjoint functors preserve limits

Left adjoint functors preserve colimits

What's at the bottom of the library?

Set plays a special role because Hom(A, B) must be a set for associativity to be defined.

```
Set: Category == with
    =: (%, %) -> Boolean

SetCategory: MathCategory Set with
                       Final Set with
                     Initial Set with
                     Product Set with
                   CoProduct Set with
```

... but these may not be sets because there can be duplicates.

$\Rightarrow$ Redefine what a "Set" means.

Domain : Category == with { }

Domain Category : Math Category Domain with

...

Axiom: Domain is a category with Initial / Final / Product / Coproduct objects.

- Model of Computation
- Don't have to worry about pre-Set theory axioms.
- Contains all needed "data structures"
- Sets come from free construction on Domain with pointer equality
- Natural bottom of the library. "The thing with all structure removed."

Recursive Aldor Categories

```
Arrow(Obj:Category):Category == with
    domain:   Obj
    codomain: Obj
    put: (domain,codomain) -> %
    get: % -> (domain,codomain)


Id(Obj:Category):Category == with
    id:  (A:Obj)->(A->A)
    default
        id(a:A):(A->A) == (a:A):A +-> a

Compose(Obj:Category):Category == with
    compose: (A:Obj,B:Obj,C:Obj) -> (A->B,B->C) -> (A->C)
    default
        compose(A:Obj,B:Obj,C:Obj)(f:A->B,g:B->C):(A->C) ==
              (a:A):C +-> g f a

MathCategory(Obj:Category):Category == Id Obj with Compose Obj
    with MathCategory Arrow Obj
```

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, dotted, "\alpha"'] & B \arrow[d, dotted, "\beta"] \\
A' \arrow[r, "f'"] & B'
\end{tikzcd}
$$

Slice Categories e.g. Fiber Bundles

```
Slice (Obj : Category, X : Obj) : Category == with
    slice : % -> X

SliceCategory (Obj : Category, X : Obj) :
    MathCategory Slice (Obj, X) with
        Final Slice (Obj, X) == add

    1 : Slice (Obj, X) == add
        Rep == X; import from Rep
        slice : % -> X == (x : X) : X +-> rep x

    1 (A : Slice (Obj, X)) : (A -> 1) ==
        (a : A) : 1 +-> slice a pretend 1
```

$$\begin{tikzcd} A \arrow[dr, "a"'] & B \arrow[d, "b"] & C \arrow[dl, "c"] \\ & X & \end{tikzcd} \qquad \begin{tikzcd} A \arrow[rr, "f"] \arrow[dr, "a"'] & & B \arrow[dl, "b"] \\ & X & \end{tikzcd}$$

Skeletal Categories

```
SkeletalCategories (P: Preorder):
    MathCategory Categorify P with
     hom: (A: Categorify P, B: Categorify P) ->
              List A -> B

== add
    hom(A: Categorify P, B: Categorify P): List A -> B ==
        input from P
        if (value $ A) <= (value $ B) then
            [ (a:A): B +-> never ]
        else
            [ ]


  Categorify(T: Type): Category == with
      value : T


   IntC : MathCategory Categorify Integer
        == SkeletalCategory Integer
```

$1 \le 2 \quad 2 \le 5 \Rightarrow 1 \le 5$

Limits

```
SI ==> SingleInteger

(Obj:Category)^(n:SI):Category == with
     dom: Tuple Obj
     put: dom -> %
     get: % -> dom

nTuple(Obj:Category,t:Tuple Obj):Obj^(length t) == add
     dom: Tuple Obj == t
     Rep == Record t
     put(x:dom):% == per [x]
     get(x:%):dom == explode rep x

Diagonal(Obj:Category,n:SI)(X:Obj):(Obj^n) ==
    nTuple (Obj,(X for i:SI in 1..n)) pretend Obj^n

Product(Obj:Category,n:SI):Category ==
         RightAdjoint(Obj  ,Obj^n,Diagonal(Obj,n)) with
CoProduct(Obj:Category,n:SI):Category ==
          LeftAdjoint(Obj^n,Obj  ,Diagonal(Obj,n)) with
```

Product is the right adjoint of the diagonal functor

$$\Delta : C \to C^n$$

CoProduct is the left adjoint of Δ.

Summary

Category theory does seem to fit in Aldor.

"Parametric polymorphism", dependent types, curried functions are essential for this.

We seem to be able to absorb theorems into the system - ideally as category defaults.

There are some limitations due to the requirement that domains be determined at compile time.

Hom vs "$\rightarrow$" styles depends a bit on current work at N.A.G. Hom style is needed for Homotopy and other applications.

The category of Domains provides a model of computation.

$$X \xrightarrow{\begin{pmatrix} f \\ f_\wedge \\ f_\vee \end{pmatrix}} Y \xrightarrow{\begin{pmatrix} g \\ g_\wedge \\ g_\vee \end{pmatrix}} Z$$

$$\begin{pmatrix} g \circ f \\ f_\wedge \circ g_\wedge \\ f_\vee \circ g_\vee \end{pmatrix}$$

$$f(f_\wedge(y)) \leq y$$

$$f(f_\vee(y)) \geq y$$