

# Algebra of Programming, Lecture 1

Jeremy Gibbons

The Algebra of Programming draws an analogy between structure of data and programs that operate on that data. The lectures will cover the following topics:

- folds and unfolds (catamorphisms and anamorphisms)
- generalizations of folds and unfolds (paramorphisms, histomorphisms, hylomorphisms, ...)
- metamorphisms that represent changes
- traversals of data structures

Fantastic Morphisms and Where to Find Them is a literature survey that talks about these different terms in more detail.

## Products and Forks

Given sets  $A$  and  $B$ , an example of an *explicit* construction would be to define

$$A \times B = \{(a, b) \mid a \in A, b \in B\}.$$

By contrast, we could instead provide the following *implicit* construction:

A **product** of  $A$  and  $B$  is a triple  $(X, fst, snd)$  where  $X$  is a set,  $fst : X \rightarrow A$ ,  $snd : X \rightarrow B$  such that for all  $C$ , given  $f : C \rightarrow A$  and  $g : C \rightarrow B$ , there exists a unique  $h : C \rightarrow X$  such that  $f = fst \circ h$  and  $g = snd \circ h$ . In other words, a product is given by this universal property it enjoys. We will see that other constructions are also constructed in this style.

In other words, if I have a triple satisfying this definition, but you provide a different triple, then mine must be at least as expressive. Hence the triple is unique up to isomorphism. We write  $(A \times B, fst, snd)$  as “the” product, although alternatives exist — for example,  $B \times A$ . When  $X = A \times B$ , our function  $h$  is simply  $h(c) = (f(c), g(c))$ .

We can express this definition as a commutative diagram:

$$\begin{array}{ccc} & A \times B & \\ & \swarrow \text{fst} & \searrow \text{snd} \\ A & \xleftarrow{f} C \xrightarrow{g} & B \\ & \uparrow h & \end{array}$$

Here the uniqueness of  $h$  is important and  $f$  and  $g$  are enough to determine a unique  $h$ . We define the notation  $\Delta$  (pronounced “fork”) and say that

$$h = f \Delta g \iff fst \circ h = f \wedge snd \circ h = g.$$

It follows from the definition that

- If  $h = f \Delta g$  is made true, the right side can be rewritten as  $fst \circ (f \Delta g) = f \wedge snd \circ (f \Delta g) = g$ .
- Conversely, if  $fst \circ h = f \wedge snd \circ h = g$  is made true, we can rewrite the left side as  $h = (fst \circ h) \Delta (snd \circ h)$ .

Looking more closely at fork,

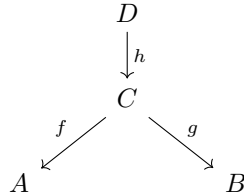
$$fst \Delta snd = id$$

since it takes the first and second of a pair and puts it together.

We can also write

$$(f \Delta g) \circ h = (f \circ h) \Delta (g \circ h),$$

which we visualize with the following diagram, for arbitrary  $A, B, C, D$ :

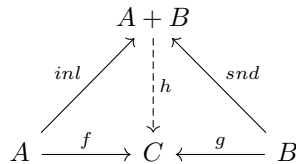


## Coproducts and Joins

A **coproduct** (or **sum**) can be explicitly defined as

$$A + B = \{(0, a) \mid a \in A\} \cup \{(1, b) \mid b \in B\}.$$

Or, given  $inl : A \rightarrow A + B$  and  $inr : B \rightarrow A + B$ , we could have the diagram



We define the notation  $\nabla$  (pronounced “join”) and say that

$$h = f \nabla g \iff h \circ inl = f \wedge h \circ inr = g.$$

## Functors

Categorically, we say that the product ( $\times$ ) and coproduct ( $+$ ) are functors. A functor  $F$  maps a type  $A$  to a type  $Fa$ , and a function  $f : A \rightarrow B$  to a function  $Ff : FA \rightarrow FB$ :

$$\frac{f : A \rightarrow B}{Ff : FA \rightarrow FB}$$

We are specifically interested in **polynomial functors**, which use  $\times, +, 1$  (the unit type, whose sole inhabitant is  $*$ ), and constants.

Examples of polynomial functors include:

- $N(X) = 1 + X$ .
- $L(X) = 1 + \mathbb{N} \times X$ .
- $T_A(X) = A + X \times X$ .

Intuitively, we can think about  $N$  as natural numbers,  $L$  as lists of natural numbers, and  $T_A$  as binary trees with leaves of type  $A$ .

We define natural numbers to be zero or the successor of another natural number, which corresponds with the definition of  $N$ . Similarly, lists are defined as an empty list or a pair of the head and tail.

## Fixed Points

The fixed points of a polynomial functor  $F$  are the types  $X$  for which  $X = FX$ . We generally write the least fixed point of a functor  $F$  as  $\mu F$ :

$$\mu F \approx F(\mu F)$$

where  $F$  describes the shape of a datatype. (Note that it is not obvious that the aforementioned functors have fixed points in categories other than  $\text{Set}$ .)

Given some functor  $F$ ,

- we choose  $\mu F$  and  $\text{in} : F(\mu F) \rightarrow \mu F$  such that,
- for all other types  $A$  and functions  $f : F(A) \rightarrow A$ ,
- there exists a unique  $h : \mu F \rightarrow A$  satisfying the universal property

$$h \circ \text{in} = f \circ Fh$$

which can be visualized in the following commutative diagram:

$$\begin{array}{ccc} F(\mu F) & \xrightarrow{Fh} & F(A) \\ \text{in} \downarrow & & \downarrow f \\ \mu F & \xrightarrow{h} & A \end{array}$$

For example, suppose we have  $f = \text{add}$ , defined below:

$$\begin{aligned} \text{add} &: (LN) \rightarrow \mathbb{N} \\ &: (1 + \mathbb{N} \times \mathbb{N}) \rightarrow \mathbb{N} \\ \text{add}(\text{inl } x) &= 0 \\ \text{add}(\text{inr } (x, y)) &= x + y \end{aligned}$$

When looking at the commutative diagram keeping lists in mind,  $F(\mu F)$  is a collection of the fragments of bits of the list.  $\text{in}$  is a constructor that gives you  $\mu F$ , from which you can get  $A$ , the sum of the list, using  $h$ . Alternatively, you can construct a list of natural numbers from  $F(\mu F)$  to get  $F(A)$  and then apply the  $\text{add}$  function to get the sum of the list.

## Folds

$h$  is called the *fold* or *cata* of  $f$  in the above commutative diagram. In particular,

$$h = \text{cata } f \iff h \circ \text{in} = f \circ Fh.$$

The Haskell exercise can be found [here](#).