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, $fst : X \to A$, $snd : X \to B$ such that for all C, given $f : C \to A$ and $g : C \to B$, there exists a unique $h : C \to X$ 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:

$$\begin{tikzcd}
 & A \times B \arrow[dl, "fst"'] \arrow[dr, "snd"] & \\
A & C \arrow[l, "f"] \arrow[r, "g"'] \arrow[u, dashed, "h"] & B
\end{tikzcd}$$

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

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

It follows from the definition that

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

Looking more closely at fork,

$$fst \vartriangle snd = id$$

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

We can also write

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

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

$$\begin{tikzcd} & D \arrow[d, "h"] & \\ & C \arrow[dl, "f"'] \arrow[dr, "g"] & \\ A & & B \end{tikzcd}$$

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 $inl : A \to A + B$ and $inr : B \to A + B$, we could have the diagram

$$\begin{tikzcd} & A + B \arrow[d, dashed, "h"] & \\ A \arrow[ur, "inl"] \arrow[r, "f"] & C & B \arrow[ul, "snd"'] \arrow[l, "g"'] \end{tikzcd}$$

We define the notation $\triangledown$ (pronounced "join") and say that

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

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 $f : A \to B$ to a function $Ff : FA \to FB$:

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

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 μF:

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

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 Set.)

Given some functor F,

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

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

which can be visualized in the following commutative diagram:

$$\begin{tikzcd} F(\mu F) \arrow[r, "Fh"] \arrow[d, "\text{in}"'] & F(A) \arrow[d, "f"] \\ \mu F \arrow[r, dashed, "h"'] & A \end{tikzcd}$$

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

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

When looking at the commutative diagram keeping lists in mind, $F(\mu F)$ is a collection of the fragments of bits of the list. *in* is a constructor that gives you μF, 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 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.