MapReduce — a two-page explanation for laymen

Maarten M. Fokkinga

Version of December 5, 2008, 16:29

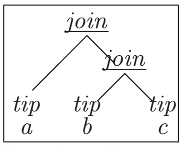
ABSTRACT

Map and *Reduce* are generic, useful notions for computing science; together they are equally expressive as simple inductive definitions over trees/lists/bags/sets.

1. Datatypes

Let A be a set. Consider the datatype of finite binary trees over A; it consists of a set T_A and two constructors *tip* and *join*:

$$\begin{array}{lll} T_A & : & \text{set} \\ tip & : & A \to T_A \\ join & : & T_A \times T_A \to T_A \end{array}$$



Function *tip* makes an A-element into a "singleton" tree, and operation *join* joins two trees into one. By definition, set T_A contains *precisely* the elements that are generated by the constructors. We'll use the notational convention that $x\ \underline{join}\ y$ stands for $join(x, y)$. So, the example tree above is denoted: $tip\,a\ \ \underline{join}\ \ (tip\,b\ \ \underline{join}\ \ tip\,c)$.

Some other useful datatypes can be obtained by further postulations:

- Postulating *associativity* of *join* (that is, $x\,\underline{join}\,(y\,\underline{join}\ z) = (x\ \underline{join}\ y)\ \underline{join}\ z$) makes the datatype into that of *lists*; the "tree structure" doesn't make sense but ordering of the tips remains significant, e.g.:

  $$\begin{array}{l} tip\,a\ \underline{join}\ (tip\,b\ \underline{join}\ (\ldots\ \underline{join}\ tip\,c)...) \\ = \\ (...(tip\,a\ \underline{join}\ tip\,b)\ \underline{join}\ \ldots)\ \underline{join}\ tip\,c \end{array}$$

  Hence the parentheses can be left out without causing semantic ambiguity. So, the list containing precisely $a, b, \ldots, c$, in that order, is denoted by the expression $tip\,a\ \underline{join}\ tip\,b\ \underline{join}\ \ldots\ \underline{join}\ tip\,c$.

- In addition postulating *commutativity* of *join* (that is, $x\ \underline{join}\ y = y\ \underline{join}\ x$) makes the datatype into that of *bags*; ordering of the tips doesn't make sense but multiplicity of tips remains significant:

  $$\begin{array}{l} tip\,a\ \underline{join}\ tip\,b\ \underline{join}\ tip\,a \\ = \\ tip\,a\ \underline{join}\ tip\,a\ \underline{join}\ tip\,b \end{array}$$

  Thus these two expressions denote the bag containing precisely a twice and b once.

- In addition postulating *absorptivity* of *join* (that is, $x\ \underline{join}\ x = x$) makes the datatype into that of *sets*; multiplicity of the tips doesn't make sense but existence (or membership) remains significant:

  $$tip\,a\ \underline{join}\ tip\,a\ \underline{join}\ tip\,b \ = \ tip\,a\ \underline{join}\ tip\,b$$

  Thus these two expressions denote the set containing precisely a and b.

- Postulating the existence in T_A of a neutral element *nil* for *join* (that is, $nil\ \underline{join}\ x = x = x\ \underline{join}\ nil$), the set T_A has an element that plays the role of "empty tree/list/bag/set", namely *nil*.

These datatypes occur frequently in computing science. Many more datatypes can be formalized along these lines; §5 discusses an alternative, "right biassed", definition of lists.

2. Catamorphism

In order to do computations over datatypes like those above, the very first thing that comes to mind is: functions defined by induction on the structure of the argument. Here is a typical example; we define a function $h : T_A \to B$, for given set B, function $f : A \to B$ and operation $\otimes : B \times B \to B$, by:

$$\begin{array}{lcll} h\,(tip\,a) & = & f\ a, & \text{for all } a : A \\ h\,(x\ \underline{join}\ y) & = & (h\,x) \otimes (h\,y), & \text{for all } x,y\!:\!T_A \end{array}$$

The effect of applying h boils down to the systematic replacement of the constructors *tip* and *join* by the items f and $\otimes$; schematically:

$$\begin{array}{rcl} x & = & \ldots\ tip\,a\ \underline{join}\ (tip\,b\ \underline{join}\ tip\,c)\ \ldots \\ h\ x & = & \ldots\ f\ \ a\ \ \otimes\ \ (\ f\ \ b\ \ \otimes\ \ \ f\ \ c)\ \ldots \end{array}$$

Let us denote the h thus defined by $\llparenthesis tip, join \to f, \otimes \rrparenthesis$ or simply $\llparenthesis f, \otimes \rrparenthesis$, and call it the *catamorphism* or *fold* determined by $f, \otimes$. In order that h is well defined, operation $\otimes$ should at least satisfy the properties of *join*; for example, if *join* is commutative, then $\otimes$ must be commutative for otherwise the definition of h might lead to a contradiction; see §4. And, if *join* has a neutral element, then so must $\otimes$, and in that case we define: $h\ nil_{join} = nil_{\otimes}$.

There exist computable total functions on these datatypes that cannot be expressed by the above simple form of induction, but nevertheless many forms of inductive definitions and indeed many algorithms can be expressed by catamorphisms in combination with other well-known constructs that are not specific for our datatypes. Some examples will be given in §4. So, this kind of inductively defined functions, which we have called *catamorphism* or *fold*, suffices for a great deal of practical algorithms.

3. MapReduce

The systematic replacement of the two items *tip* and *join* by the two items f and $\otimes$, respectively, has been denoted by one single expression: $\llparenthesis f, \otimes \rrparenthesis$. However, it makes sense to do the replacements separately. So, let us tear them apart and define an "f-map" (denoted $f*$) that applies f inside every tip, e.g.:

$$\begin{aligned} f*\,(\,\mathit{tip}\ a\ \underline{\mathit{join}}\,(\mathit{tip}\ b\ \underline{\mathit{join}}\ \mathit{tip}\ c\,)\,) &= \\ \mathit{tip}\,fa\ \underline{\mathit{join}}\,(\mathit{tip}\,fb\ \underline{\mathit{join}}\ \mathit{tip}\,fc) & \end{aligned}$$

and a "$\otimes$-reduce" (denoted $\otimes/$) that aggregates all tip values, using $\otimes$ at every *join*-node:

$$\begin{aligned} \otimes/\ (\,\mathit{tip}\ a\ \underline{\mathit{join}}\,(\mathit{tip}\ b\ \underline{\mathit{join}}\ \mathit{tip}\ c)\,) &= \\ a \quad \otimes \quad (\quad b \quad \otimes \quad c \quad) & \end{aligned}$$

Here are the definitions, using $\circ$ for function composition, $(f{\circ}g)x = f(g\,x)$, and *id* for identity, $\mathit{id}\,x = x$; assuming $f : A{\rightarrow}B$ and $\otimes : B{\times}B{\rightarrow}B$:

$$\begin{aligned} f* &= \llparenthesis \mathit{tip} \circ f,\ \mathit{join} \rrparenthesis && : T_A \rightarrow T_B \\ \otimes/ &= \llparenthesis \mathit{id},\ \otimes \rrparenthesis && : T_B \rightarrow B \end{aligned}$$

One can prove (e.g., by systematic replacements):

$$\llparenthesis f,\ \otimes \rrparenthesis \;=\; \otimes/ \;\circ\; f*$$

These three equations show that maps and reduces together are equally expressive as catamorphisms. Hence, maps and reduces (together with other operations not specific for our datatypes) suffice for a great deal of practical algorithms on our datatypes.

4. Examples

The number of tips ("size") of a tree/list/bag is computed by taking 1 at each tip and aggregating (summing, in this case) these numbers at each node:

$$\mathit{size} \;=\; \llparenthesis f,\ + \rrparenthesis \;=\; +/ \circ f* \qquad \text{where } f\,a{=}1$$

Since $+$ is not absorptive, *size* is not defined when *join* is absorptive, i.e., for sets. (Applying *size* to absorptive *join* gives the contradiction: $1{+}1 = \mathit{size}(\mathit{tip}\,a\ \underline{\mathit{join}}\ \mathit{tip}\,a) = \mathit{size}(\mathit{tip}\,a) = 1$.)

Merge sort. Let us first define an aggregation operation *merge* that merges two *sorted* trees/lists/bags/sets into a single *sorted* one:

$$\begin{aligned} &x\ \underline{\mathit{merge}}\ \mathit{nil} \;=\; x \\ &\mathit{nil}\ \underline{\mathit{merge}}\ x \;=\; x \\ &(\mathit{tip}\,a\ \underline{\mathit{join}}\ x)\ \underline{\mathit{merge}}\ (\mathit{tip}\,b\ \underline{\mathit{join}}\ y) \\ &\quad = \mathit{tip}\,a\ \underline{\mathit{join}}\ (x\ \underline{\mathit{merge}}\ (\mathit{tip}\,b\ \underline{\mathit{join}}\ y)), \quad \text{if } a{\le}b \\ &\quad = \mathit{tip}\,b\ \underline{\mathit{join}}\ ((\mathit{tip}\,a\ \underline{\mathit{join}}\ x)\ \underline{\mathit{merge}}\ y), \quad \text{if } a{\ge}b \end{aligned}$$

Now, sorting is just "aggregation by *merge*", after making each element into a singleton:

$$\mathit{sort} \;=\; \llparenthesis \mathit{tip},\ \mathit{merge} \rrparenthesis \;=\; \mathit{merge}/ \circ \mathit{tip}*$$

5. Alternative definition of lists

Suppose we define the datatype of lists in the following "right biassed" way:

$$\begin{aligned} L_A &: \text{set} \\ \mathit{nil} &: L_A \\ \mathit{cons} &: A \times L_A \rightarrow L_A \end{aligned}$$

An example list is: $a\ \underline{\mathit{cons}}\ (b\ \underline{\mathit{cons}}\ (c\ \underline{\mathit{cons}}\ \mathit{nil}))$. Here too, the notion of catamorphism makes sense; it is a function h defined by induction on the structure of its argument, and is completely determined by a given set B and element $e : B$ and operation $\otimes : A \times B \rightarrow B$:

$$\begin{aligned} x &= a\ \underline{\mathit{cons}}\ (b\ \underline{\mathit{cons}} \ldots (c\ \underline{\mathit{cons}}\ \mathit{nil})) \\ h\,x &= a \ \otimes \ (b \ \otimes \ \ldots (c \ \otimes \ e\ )) \end{aligned}$$

This catamorphism h is sometimes called the right-reduce or *foldr* determined by $\otimes$ and e. Taking $a \otimes x = f\,a\ \underline{\mathit{cons}}\ x$ and $e = \mathit{nil}$ it turns out that the h thus defined is "f-map" in the sense that it applies f to every tip value in the list.

6. Closing remarks

(1) Maps, and to a lesser extent reduces, are suitable for distributed evaluation. (2) For all inductively defined datatypes the notions of catamorphism and map make sense, but for some the notion of reduce is problematic. (3) Algorithmics is the field where these ideas are put into practice to derive efficient algorithms; category theory is the field where the notions themselves are studied (datatype, catamorphism, map, and reduce).

7. Sources

The tree/list/bag/set approach has been proposed in 1981 by Boom [3]. Meertens [9] showed the equivalence of catamorphisms with maps-and-reduces. This led Bird to exploit these notions for a Theory of Lists [1]; he had many followers, for example [7], and culminated in the master piece [2]. Malcolm [8] invented the banana brackets $\llparenthesis\ \rrparenthesis$. Our paper [10] describes further concepts analogous to catamorphisms, with corresponding symbols: lenses, envelopes, barbed wire. I myself have studied the categorical approach to data types [6], and the presence of equations (laws) [5]. Recently, Google [4] has applied maps and reduces to execute programs on a cluster of machines.

REFERENCES

[1] R.S. Bird. An Introduction to the Theory of Lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*, pages 3–42. Springer Verlag, 1987.

[2] R.S. Bird and O. de Moor. *Algebra of Programming*. Prentice-Hall International, 1996.

[3] H.J. Boom. Further thoughts on Abstracto. Working paper ELC-9, IFIP WG 2.1, 1981.

[4] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI'04*, pages 137–150. USENIX, 2004.

[5] M.M. Fokkinga. Datatype laws without signatures. *Math. Structures in Computer Science*, 6(1):1–32, 1996.

[6] M.M. Fokkinga. *Law and Order in Algorithmics*. PhD thesis, University of Twente, The Netherlands, 1992.

[7] Graham Hutton. A tutorial on the universality and expressiveness of fold. *J. Functional Programming, Cambridge University Press*, 9(4):355–372, 1999.

[8] G. Malcolm. *Algebraic Data Types and Program Transformation*. PhD thesis, University of Groningen, The Netherlands, 1990.

[9] L. Meertens. Algorithmics — towards programming as a mathematical activity. In J.W. de Bakker and J.C. van Vliet, eds., *Proc. CWI Symp. on Math. and Comp. Sc.*, pages 289–334. North-Holland, 1986.

[10] E. Meijer, M.M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In *FPCA91: Functional Programming Languages and Computer Architecture*, volume 523 of *Lect. Notes in Comp. Sc.*, pages 124–144. Springer Verlag, 1991.