Upwards and downwards accumulations on trees

JEREMY GIBBONS

ABSTRACT. An *accumulation* is a higher-order operation over structured objects of some type; it leaves the 'shape' of an object unchanged, but replaces each element of that object with some accumulated information about the other elements. Upwards and downwards accumulations on trees are two instances of this scheme; they replace each element of a tree with some function—in fact, some homomorphism—of that element's descendants and of its ancestors, respectively. These two operations can be thought of as passing information up and down the tree. We describe these two accumulations, and show how together they solve the so-called 'prefix sums' problem.

1 Introduction

The value of being able to calculate computer programs formally from their specifications is now widely recognized. This ability to calculate can only be achieved with the aid of mathematically precise and concise tools. In this paper we look at the calculation of solutions to problems about trees. Trees are important in computing because they capture the idea of hierarchical structure. They permit fast parallel collection and dissemination of information among their elements; indeed, it could be argued that *all* algorithms that take logarithmic time, whether sequentially or in parallel, do so because of an underlying tree structure.

We introduce two tools for reasoning with problems about trees, namely, *upwards* and *downwards accumulations* on trees. These accumulations embody the notions of passing information up a tree, from the leaves towards the root, and down, from the root towards the leaves. We use these accumulations in the derivation of a fast algorithm for the prefix sums problem.

The workbench on which we use these tools is the *Bird-Meertens formalism* (Meertens, 1986; Bird, 1987, 1988; Backhouse, 1989). We give a crash course in the relevant notation below.

Author's address: Dept of Computer Science, University of Auckland, Private Bag 92019, Auckland, New Zealand. Email: `jeremy@cs.aukuni.ac.nz`. This is a revised version of the paper appearing in *Mathematics of Program Construction* (Springer Lecture Notes in Computer Science, 1992).

2 Notation

The identity function is written id; the constant function always returning a is written $!\mathsf{a}$. Function application is written with an infix $\cdot$, so $!\mathsf{a}\cdot\mathsf{b} = \mathsf{a}$. Application is tightest binding, and *right* associative, so $\mathsf{f}\cdot\mathsf{g}\cdot\mathsf{a}$ parses as $\mathsf{f}\cdot(\mathsf{g}\cdot\mathsf{a})$; we find this more useful than left associative application. Function composition is backwards, written with an infix $\circ$, and is weakest binding:

$$(\mathsf{f} \circ \mathsf{g})\cdot\mathsf{a} \;=\; \mathsf{f}\cdot\mathsf{g}\cdot\mathsf{a}$$

The Bird-Meertens formalism makes free use of infix binary operators. Such operators are turned into unary functions by *sectioning*:

$$\langle\mathsf{a}\oplus\rangle\cdot\mathsf{b} \;=\; \mathsf{a}\oplus\mathsf{b} \;=\; \langle\oplus\mathsf{b}\rangle\cdot\mathsf{a}$$

The *converse* of a binary operator $\oplus$ is written $\widetilde{\oplus}$ and satisfies

$$\mathsf{a}\,\widetilde{\oplus}\,\mathsf{b} \;=\; \mathsf{b}\oplus\mathsf{a}$$

The type judgement 'a has type A' is written $\mathsf{a}\in\mathsf{A}$ (this is not intended to mean that types are sets). The function type former is written $\rightarrow$. The cartesian product of two types is denoted by $\|$, and their sum by $|$. Indeed, $\|$ and $|$ are bifunctors, acting on functions as well as types: if $\mathsf{f}\in\mathsf{A}\rightarrow\mathsf{C}$ and $\mathsf{g}\in\mathsf{B}\rightarrow\mathsf{D}$ then

$$\begin{array}{lcl} \mathsf{f}\,\|\,\mathsf{g} & \in & \mathsf{A}\,\|\,\mathsf{B} \rightarrow \mathsf{C}\,\|\,\mathsf{D} \\ \mathsf{f}\,|\,\mathsf{g} & \in & \mathsf{A}\,|\,\mathsf{B} \rightarrow \mathsf{C}\,|\,\mathsf{D} \end{array}$$

We write A^2 and f^2 as abbreviations for $\mathsf{A}\,\|\,\mathsf{A}$ and $\mathsf{f}\,\|\,\mathsf{f}$. The product and sum morphisms 'fork' and 'join' are denoted by $\curlywedge$ and $\curlyvee$, the shapes suggesting processes splitting and recombining as they 'move down the page'; if $\mathsf{f}\in\mathsf{A}\rightarrow\mathsf{B}$ and $\mathsf{g}\in\mathsf{A}\rightarrow\mathsf{C}$, and $\mathsf{h}\in\mathsf{B}\rightarrow\mathsf{A}$ and $\mathsf{k}\in\mathsf{C}\rightarrow\mathsf{A}$, then

$$\begin{array}{lcl} \mathsf{f}\curlywedge\mathsf{g} & \in & \mathsf{A} \rightarrow \mathsf{B}\,\|\,\mathsf{C} \\ \mathsf{h}\curlyvee\mathsf{k} & \in & \mathsf{B}\,|\,\mathsf{C} \rightarrow \mathsf{A} \end{array}$$

The projections for products are

$$\begin{array}{lcl} \ll & \in & \mathsf{A}\,\|\,\mathsf{B} \rightarrow \mathsf{A} \\ \gg & \in & \mathsf{A}\,\|\,\mathsf{B} \rightarrow \mathsf{B} \end{array}$$

We do not need injections for sums in this paper.

Data types are constructed as the least fixed points of polynomial functors, that is, as the initial algebras in the appropriate categories of algebras. We will just use the ideas informally here, not having the space to present them formally; the reader is referred to Malcolm (1990) or Hagino (1987) for the details. For functor F, an F-algebra is a pair (X,τ) such that $\tau\in\mathsf{F}\cdot\mathsf{X}\rightarrow\mathsf{X}$. The trees we will be considering, *homogeneous binary trees*, are defined by the equation

$$\mathsf{tree}\cdot\mathsf{A} \;=\; \vartriangle\cdot\mathsf{A} \;|\; \mathsf{tree}\cdot\mathsf{A} \mathbin{\pm_{\mathsf{A}}} \mathsf{tree}\cdot\mathsf{A}$$

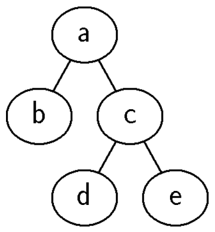
Informally, this says that the algebra $(\mathsf{tree}\cdot\mathsf{A}, \triangle \curlyvee \text{木})$ of trees with elements of type A is the least fixed point of the functor T that maps X to $\mathsf{A} \mid (\mathsf{X} \parallel \mathsf{A} \parallel \mathsf{X})$; the constructors $\triangle$ (pronounced 'leaf') and 木 (a corruption of the Chinese ideogram 木, pronounced 'moo' and meaning 'wood' or 'tree') have types

$$\begin{aligned} \triangle &\in \mathsf{A} \to \mathsf{tree}\cdot\mathsf{A} \\ \text{木} &\in \mathsf{tree}\cdot\mathsf{A} \parallel \mathsf{A} \parallel \mathsf{tree}\cdot\mathsf{A} \to \mathsf{tree}\cdot\mathsf{A} \end{aligned}$$

Note that 木 is a *ternary* operator; its middle argument is written as a subscript. For example, the tree

$$\triangle\cdot\mathsf{b} \ \text{木}_\mathsf{a} \ (\triangle\cdot\mathsf{d} \ \text{木}_\mathsf{c} \ \triangle\cdot\mathsf{e})$$

corresponds to the tree



We will call this tree five, and use it as a running example.

The operation on types that sends A to $\mathsf{tree}\cdot\mathsf{A}$ is a functor: its action on functions, written with a postfix '$*$', respects identity and composition. Thus, if $\mathsf{f} \in \mathsf{A} \to \mathsf{B}$, then $\mathsf{f}* \in \mathsf{tree}\cdot\mathsf{A} \to \mathsf{tree}\cdot\mathsf{B}$, and $\mathsf{id}* = \mathsf{id}$ and $(\mathsf{f} \circ \mathsf{g})* = \mathsf{f}* \circ \mathsf{g}*$.

We say that a function h is (f, g) F-*promotable* if

$$\mathsf{h} \circ \mathsf{f} \ = \ \mathsf{g} \circ \mathsf{F}\cdot\mathsf{h}$$

The important fact about the initial algebra in a category of algebras, the *unique extension property*, states that, if (X, τ) is the initial F-algebra, then for a given f there is a *unique* function that is (τ, f) F-promotable. This function is called a *catamorphism* and is written $(\!|\mathsf{F}\colon \mathsf{f}|\!)$; if the functor F is clear from context, we write simply $(\!|\mathsf{f}|\!)$. In the case of trees, the catamorphism $(\!|\mathsf{f} \curlyvee \odot|\!)$ satisfies

$$\begin{aligned} (\!|\mathsf{f} \curlyvee \odot|\!)\cdot\triangle\cdot\mathsf{a} &= \mathsf{f}\cdot\mathsf{a} \\ (\!|\mathsf{f} \curlyvee \odot|\!)\cdot(\mathsf{x} \ \text{木}_\mathsf{a} \ \mathsf{y}) &= (\!|\mathsf{f} \curlyvee \odot|\!)\cdot\mathsf{x} \ \odot_\mathsf{a} \ (\!|\mathsf{f} \curlyvee \odot|\!)\cdot\mathsf{y} \end{aligned}$$

and is the unique function that is $(\triangle \curlyvee \text{木}, \mathsf{f} \curlyvee \odot)$ T-promotable. (Note that h is $(\tau_1 \curlyvee \cdots \curlyvee \tau_\mathsf{n}, \mathsf{f}_1 \curlyvee \cdots \curlyvee \mathsf{f}_\mathsf{n})$ F-promotable if $\mathsf{F}\cdot\mathsf{X} = \mathsf{F}_1\cdot\mathsf{X} \mid \cdots \mid \mathsf{F}_\mathsf{n}\cdot\mathsf{X}$, and h is $(\tau_\mathsf{i}, \mathsf{f}_\mathsf{i})$ F_i-promotable for $1 \le \mathsf{i} \le \mathsf{n}$.) For example, the functions returning the numbers of leaves and branches of a tree are catamorphisms, given by

$$\begin{aligned} \mathsf{leaves} &= (\!|!1 \curlyvee \oplus|\!) \qquad \text{where} \quad \mathsf{u} \oplus_\mathsf{a} \mathsf{v} = \mathsf{u} + \mathsf{v} \\ \mathsf{branches} &= (\!|!0 \curlyvee \otimes|\!) \qquad \text{where} \quad \mathsf{u} \otimes_\mathsf{a} \mathsf{v} = \mathsf{u} + 1 + \mathsf{v} \end{aligned}$$

Another example is the function root, which returns the root of a tree:

$$\mathsf{root} \;=\; (\!\![\mathsf{id} \curlyvee \odot]\!\!) \qquad \text{where} \quad \mathsf{u} \odot_{\mathsf{a}} \mathsf{v} = \mathsf{a}$$

and another is the identity function, the catamorphism built from the constructors of the type:

$$\mathsf{id} \;=\; (\!\![\vartriangle \curlyvee \underset{\smile}{+}]\!\!)$$

Catamorphisms are important because they are 'eminently manipulable'.

One corollary of the unique extension property is called the *promotion theorem*; the proof is in Malcolm's thesis, among other places.

THEOREM (1) If h is (f, g) F-promotable, then $\mathsf{h} \circ (\!\![\mathsf{F}\colon \mathsf{f}]\!\!) = (\!\![\mathsf{F}\colon \mathsf{g}]\!\!)$. $\Diamond$

For example, the popular student exercise in structural induction of showing that

$$\langle 1+ \rangle \circ \mathsf{branches} \;=\; \mathsf{leaves}$$

follows from the promotion theorem, because $\langle 1+ \rangle$ is $(!0 \curlyvee \otimes, !1 \curlyvee \oplus)$ T-promotable.

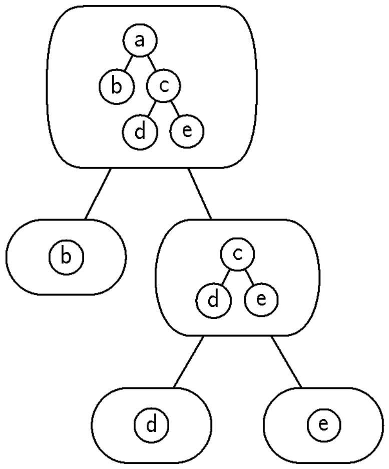
3 Upwards accumulations

We now turn to the main topic of this paper, namely accumulations. Functional programmers will be familiar with the function inits on (non-empty) lists, which returns a list of all the initial segments of its argument; for example,

$$\mathsf{inits} \cdot [\mathsf{a}, \mathsf{b}, \mathsf{c}] \;=\; [[\mathsf{a}], [\mathsf{a}, \mathsf{b}], [\mathsf{a}, \mathsf{b}, \mathsf{c}]]$$

Upwards and downwards accumulations arise from generalizing this concept to trees.

One way of thinking about inits is as replacing every element of a list with its *predecessors*, that is, with the initial segment of the list ending with that element. By analogy, the function $\mathsf{subtrees}$ on trees replaces every element of a tree with its *descendants*, that is, with the subtree rooted at that element. For example, applying $\mathsf{subtrees}$ to the tree five yields the tree of trees



The equations characterizing $\mathsf{subtrees}$ are

$$\begin{aligned} \mathsf{subtrees}\cdot\triangle\cdot\mathsf{a} &= \triangle\cdot\triangle\cdot\mathsf{a} \\ \mathsf{subtrees}\cdot(\mathsf{x} \mathbin{\underset{\smile}{+}}_{\mathsf{a}} \mathsf{y}) &= \mathsf{subtrees}\cdot\mathsf{x} \mathbin{\underset{\smile}{+}}_{\mathsf{x} \underset{\smile}{+}_{\mathsf{a}} \mathsf{y}} \mathsf{subtrees}\cdot\mathsf{y} \end{aligned}$$

We can see by case analysis that

$$\mathsf{root} \circ \mathsf{subtrees} = \mathsf{id}$$

and so

$$\mathsf{subtrees}\cdot(\mathsf{x} \mathbin{\underset{\smile}{+}}_{\mathsf{a}} \mathsf{y}) = \mathsf{subtrees}\cdot\mathsf{x} \oplus_{\mathsf{a}} \mathsf{subtrees}\cdot\mathsf{y}$$

where

$$\mathsf{u} \oplus_{\mathsf{a}} \mathsf{v} = \mathsf{u} \mathbin{\underset{\smile}{+}}_{\mathsf{z}} \mathsf{v} \qquad \text{where} \quad \mathsf{z} = \mathsf{root}\cdot\mathsf{u} \mathbin{\underset{\smile}{+}}_{\mathsf{a}} \mathsf{root}\cdot\mathsf{v}$$

That is, $\mathsf{subtrees}$ is a catamorphism, $(\!|(\triangle \circ \triangle) \curlyvee \oplus|\!)$.

Functions that 'pass information upwards', from the leaves of a tree towards the root, are characterized in terms of $\mathsf{subtrees}$:

DEFINITION (2) Functions of the form $\mathsf{g}{*} \circ \mathsf{subtrees}$ are called *upwards passes.*

◊

Suppose that h is an upwards pass, and that $\mathsf{h} = \mathsf{g}{*} \circ \mathsf{subtrees}$, so that

$$\begin{aligned} \mathsf{h}\cdot\triangle\cdot\mathsf{a} &= \triangle\cdot\mathsf{g}\cdot\triangle\cdot\mathsf{a} \\ \mathsf{h}\cdot(\mathsf{x} \mathbin{\underset{\smile}{+}}_{\mathsf{a}} \mathsf{y}) &= \mathsf{h}\cdot\mathsf{x} \mathbin{\underset{\smile}{+}}_{\mathsf{b}} \mathsf{h}\cdot\mathsf{y} \qquad \text{where} \quad \mathsf{b} = \mathsf{g}\cdot(\mathsf{x} \mathbin{\underset{\smile}{+}}_{\mathsf{a}} \mathsf{y}) \end{aligned}$$

This does not yield a quick way of computing h, even if we have one for g; for example, if g takes time proportional to the depth of the tree, then h will take parallel time proportional to the square of the depth.

However, suppose further that g is a tree catamorphism, so that

$$\mathsf{g}\cdot(\mathsf{x} \mathbin{\underset{\smile}{+}}_{\mathsf{a}} \mathsf{y}) = \mathsf{g}\cdot\mathsf{x} \odot_{\mathsf{a}} \mathsf{g}\cdot\mathsf{y}$$

for some $\odot$. Now, we already have

$$\mathsf{root} \circ \mathsf{h} \;=\; \mathsf{g}$$

and so

$$\mathsf{g} \cdot (\mathsf{x} \curlywedge_{\mathsf{a}} \mathsf{y}) \;=\; \mathsf{root} \cdot \mathsf{h} \cdot \mathsf{x} \odot_{\mathsf{a}} \mathsf{root} \cdot \mathsf{h} \cdot \mathsf{y}$$

Thus,

$$\mathsf{h} \cdot (\mathsf{x} \curlywedge_{\mathsf{a}} \mathsf{y}) \;=\; \mathsf{h} \cdot \mathsf{x} \oplus_{\mathsf{a}} \mathsf{h} \cdot \mathsf{y}$$

where

$$\mathsf{u} \oplus_{\mathsf{a}} \mathsf{v} \;=\; \mathsf{u} \curlywedge_{\mathsf{b}} \mathsf{v} \qquad \text{where} \quad \mathsf{b} = \mathsf{root} \cdot \mathsf{u} \odot_{\mathsf{a}} \mathsf{root} \cdot \mathsf{v}$$

The important point is that $\mathsf{h} \cdot (\mathsf{x} \curlywedge_{\mathsf{a}} \mathsf{y})$ can be computed from $\mathsf{h} \cdot \mathsf{x}$ and $\mathsf{h} \cdot \mathsf{y}$ using only *one* more application of $\odot$, and so h can be computed in parallel time proportional to the depth of the tree times the time taken by $\odot$. Such functions are what we mean by *upwards accumulations*:

DEFINITION (3) Functions of the form $\llbracket \mathsf{f} \curlyvee \odot \rrbracket * \circ \mathsf{subtrees}$ are called *upwards accumulations*, and are written $(\mathsf{f}, \odot)\Uparrow$. ◊

The phrase above about the time taken to compute h is rather unwieldy; we shall say instead that a function on trees can be computed *quickly up to* $\odot$, or just *quickly* when the $\odot$ is understood, if it can be computed in parallel time proportional to the depth of the tree times the time taken by $\odot$.

One simple example of an upwards accumulation is the function sizes, which replaces every element of a tree with the number of descendants it has:

$$\mathsf{sizes} \;=\; \mathsf{size} * \circ \mathsf{subtrees}$$

Since size is a catamorphism,

$$\mathsf{size} \;=\; \llbracket !1 \curlyvee \odot \rrbracket \qquad \text{where} \quad \mathsf{u} \odot_{\mathsf{a}} \mathsf{v} = \mathsf{u} + 1 + \mathsf{v}$$

we have a quick algorithm

$$\mathsf{sizes} \;=\; (!1, \odot)\Uparrow$$

for sizes.

The equation

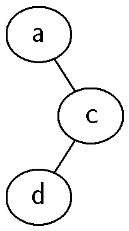
$$\llbracket \mathsf{f} \curlyvee \odot \rrbracket * \circ \mathsf{subtrees} \;=\; (\mathsf{f}, \odot)\Uparrow$$

can be seen as an efficiency-improving transformation, when used from left to right. It can also, of course, be used from right to left, when it forms a 'manipulability-improving' transformation; catamorphisms, maps and $\mathsf{subtrees}$ enjoy many useful properties, and the left hand side may be more amenable to calculation than the right. This choice between manipulability and efficiency is a characteristic of accumulations.

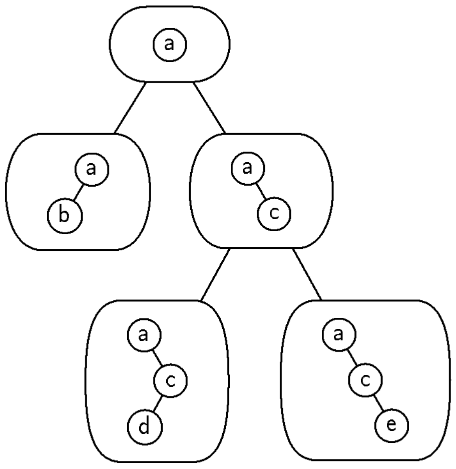
4 Downwards accumulations

We have just discussed upwards accumulation, which captures the notion of passing information up through a tree from the leaves towards the root. We turn now to *downwards* accumulation, which corresponds to passing information in the opposite direction, from the root towards the leaves.

As upwards accumulations arose by considering the function $\mathsf{subtrees}$, which replaces every element of a tree with its descendants, so downwards accumulations arise by considering the function paths, which replaces every element of a tree with its *ancestors*. The ancestors of an element in a tree themselves form a special kind of tree, called a *thread*: a tall thin tree with that element as its one and only leaf. For example, the ancestors of the element d in the tree five form the three-element thread



and applying paths to five yields the tree of threads



As a type, threads are the least fixed point of the functor H which sends X to $\mathsf{A} \mid (\mathsf{X} \parallel \mathsf{A}) \mid (\mathsf{X} \parallel \mathsf{A})$; they are given by the type equation

$$\mathsf{thread}\cdot\mathsf{A} \;=\; \diamond\cdot\mathsf{A} \mid \mathsf{thread}\cdot\mathsf{A} \swarrow \mathsf{A} \mid \mathsf{thread}\cdot\mathsf{A} \searrow \mathsf{A}$$

and the three-element thread given above is written $(\diamond\cdot\mathsf{a} \searrow \mathsf{c}) \swarrow \mathsf{d}$. The constructors $\swarrow$ and $\searrow$ could be pronounced 'left snoc' and 'right snoc', and threads thought of as 'snoc' lists with two different colours of constructor.

Now, paths is a catamorphism, and is given by the equations

$$\begin{aligned}\mathsf{paths}\cdot\triangle\cdot\mathsf{a} &= \triangle\cdot\diamond\cdot\mathsf{a}\\ \mathsf{paths}\cdot(\mathsf{x} \perp_{\mathsf{a}} \mathsf{y}) &= \langle \mathsf{a}\oslash\rangle{*}\cdot\mathsf{paths}\cdot\mathsf{x} \perp_{\diamond\cdot\mathsf{a}} \langle\mathsf{a}\obslash\rangle{*}\cdot\mathsf{paths}\cdot\mathsf{y}\end{aligned} \tag{i}$$

where

$$\begin{aligned}\mathsf{a}\oslash\mathsf{p} &= \llparenthesis \mathsf{H}\colon \langle\diamond\cdot\mathsf{a}\swarrow\rangle \curlyvee \swarrow \curlyvee \searrow\rrparenthesis\cdot\mathsf{p}\\ \mathsf{a}\obslash\mathsf{p} &= \llparenthesis \mathsf{H}\colon \langle\diamond\cdot\mathsf{a}\searrow\rangle \curlyvee \swarrow \curlyvee \searrow\rrparenthesis\cdot\mathsf{p}\end{aligned}$$

Informally, $\oslash$ and $\obslash$ ‘cons’ elements to threads; for example,

$$\mathsf{a}\obslash(\diamond\cdot\mathsf{c}\swarrow\mathsf{d}) \quad = \quad (\diamond\cdot\mathsf{a}\searrow\mathsf{c})\swarrow\mathsf{d}$$

Because threads have three constructors, thread catamorphisms involve a ‘three-way join’, which should be considered a ternary operator rather than two applications of a binary one.

Functions that ‘pass information downwards’ are characterized by the following definition.

Definition (4) Functions of the form $\mathsf{g}{*}\circ\mathsf{paths}$ are called *downwards passes*. $\diamondsuit$

As before, downwards passes need not be quick; however, if the ‘multiplier’ g is a thread catamorphism then $\mathsf{g}{*}\circ\mathsf{paths}$ is more tractable.

Definition (5) Functions of the form $\llparenthesis\mathsf{H}\colon \mathsf{f}\curlyvee\oplus\curlyvee\otimes\rrparenthesis{*}\circ\mathsf{paths}$ are called *downwards accumulations* and are written $(\mathsf{f},\oplus,\otimes)\Downarrow$. $\diamondsuit$

The downwards accumulation $(\mathsf{f},\oplus,\otimes)\Downarrow$ can be computed *quickly up to* $\oplus$ *and* $\otimes$, that is, quickly up to the more expensive of $\oplus$ and $\otimes$. To see this, we note first that a thread catamorphism composed with $\langle\mathsf{a}\oslash\rangle$ or $\langle\mathsf{a}\obslash\rangle$ is another catamorphism:

Lemma (6)

$$\begin{aligned}\llparenthesis\mathsf{H}\colon \mathsf{f}\curlyvee\oplus\curlyvee\otimes\rrparenthesis\circ\langle\mathsf{a}\oslash\rangle &= \llparenthesis\mathsf{H}\colon \langle\mathsf{f}\cdot\mathsf{a}\oplus\rangle\curlyvee\oplus\curlyvee\otimes\rrparenthesis\\ \llparenthesis\mathsf{H}\colon \mathsf{f}\curlyvee\oplus\curlyvee\otimes\rrparenthesis\circ\langle\mathsf{a}\obslash\rangle &= \llparenthesis\mathsf{H}\colon \langle\mathsf{f}\cdot\mathsf{a}\otimes\rangle\curlyvee\oplus\curlyvee\otimes\rrparenthesis\end{aligned}$$

$\diamondsuit$

Proof The catamorphism $\llparenthesis\mathsf{f}\curlyvee\oplus\curlyvee\otimes\rrparenthesis$ is $(\swarrow,\oplus)$ and $(\searrow,\otimes)$ promotable, and so it is $(\langle\diamond\cdot\mathsf{a}\swarrow\rangle\curlyvee\swarrow\curlyvee\searrow, (\llparenthesis\mathsf{f}\curlyvee\oplus\curlyvee\otimes\rrparenthesis\circ\langle\diamond\cdot\mathsf{a}\swarrow\rangle)\curlyvee\oplus\curlyvee\otimes)$ H-promotable; hence

$$\llparenthesis\mathsf{f}\curlyvee\oplus\curlyvee\otimes\rrparenthesis\circ\langle\mathsf{a}\oslash\rangle \quad = \quad \llparenthesis(\llparenthesis\mathsf{f}\curlyvee\oplus\curlyvee\otimes\rrparenthesis\circ\langle\diamond\cdot\mathsf{a}\swarrow\rangle)\curlyvee\oplus\curlyvee\otimes\rrparenthesis$$

Since $\llparenthesis\mathsf{f}\curlyvee\oplus\curlyvee\otimes\rrparenthesis\circ\langle\diamond\cdot\mathsf{a}\swarrow\rangle = \langle\mathsf{f}\cdot\mathsf{a}\oplus\rangle$, this gives us the first equation. The second is similar. $\heartsuit$

Then

$$
\begin{array}{cl}
 & (\mathsf{f},\oplus,\otimes)\Downarrow\cdot(\mathsf{x}\ \underline{\curlywedge}_{\mathsf{a}}\ \mathsf{y}) \\
= & \qquad \{\ \Downarrow\ \} \\
 & [\![\mathsf{f}\curlyvee\oplus\curlyvee\otimes]\!]*\cdot\mathsf{paths}\cdot(\mathsf{x}\ \underline{\curlywedge}_{\mathsf{a}}\ \mathsf{y}) \\
= & \qquad \{\ \mathsf{paths}\ \} \\
 & [\![\mathsf{f}\curlyvee\oplus\curlyvee\otimes]\!]*\cdot(\langle\mathsf{a}\oslash\rangle*\cdot\mathsf{paths}\cdot\mathsf{x}\ \ \underline{\curlywedge}_{\diamond\cdot\mathsf{a}}\ \langle\mathsf{a}\obslash\rangle*\cdot\mathsf{paths}\cdot\mathsf{y}) \\
= & \qquad \{\ \text{Lemma 6}\ \} \\
 & [\![\langle\mathsf{f}\cdot\mathsf{a}\oplus\rangle\curlyvee\oplus\curlyvee\otimes]\!]*\cdot\mathsf{paths}\cdot\mathsf{x}\ \ \underline{\curlywedge}_{\mathsf{f}\cdot\mathsf{a}}\ [\![\langle\mathsf{f}\cdot\mathsf{a}\otimes\rangle\curlyvee\oplus\curlyvee\otimes]\!]*\cdot\mathsf{paths}\cdot\mathsf{y} \\
= & \qquad \{\ \Downarrow\ \} \\
 & (\langle\mathsf{f}\cdot\mathsf{a}\oplus\rangle,\oplus,\otimes)\Downarrow\cdot\mathsf{x}\ \ \underline{\curlywedge}_{\mathsf{f}\cdot\mathsf{a}}\ (\langle\mathsf{f}\cdot\mathsf{a}\otimes\rangle,\oplus,\otimes)\Downarrow\cdot\mathsf{y}
\end{array}
$$

and the recursion on paths can be made in parallel after only one more application each of $\oplus$ and $\otimes$. For example, we see that paths can be computed quickly up to $\swarrow$ and $\searrow$, that is, in parallel time proportional to the depth of the tree, because paths is itself a downwards accumulation:

$$
\begin{array}{cl}
 & \mathsf{paths} \\
= & \qquad \{\ [\![\diamond\curlyvee\swarrow\curlyvee\searrow]\!]=\mathsf{id}\ \} \\
 & [\![\diamond\curlyvee\swarrow\curlyvee\searrow]\!]*\circ\mathsf{paths} \\
= & \qquad \{\ \Downarrow\ \} \\
 & (\diamond,\swarrow,\searrow)\Downarrow
\end{array}
$$

Notice, however, that downwards accumulations are not in general catamorphic: the accumulation $(\mathsf{f},\oplus,\otimes)\Downarrow$ of a branch $\mathsf{x}\ \underline{\curlywedge}_{\mathsf{a}}\ \mathsf{y}$ depends on different accumulations $(\langle\mathsf{f}\cdot\mathsf{a}\oplus\rangle,\oplus,\otimes)\Downarrow$ and $(\langle\mathsf{f}\cdot\mathsf{a}\otimes\rangle,\oplus,\otimes)\Downarrow$ of the children.

The characterization (i) we gave for paths is an instance of the frequently-occurring idiom

$$
\begin{array}{rcl}
\mathsf{h}\cdot\triangle\cdot\mathsf{a} & = & \triangle\cdot\mathsf{f}\cdot\mathsf{a} \\
\mathsf{h}\cdot(\mathsf{x}\ \underline{\curlywedge}_{\mathsf{a}}\ \mathsf{y}) & = & \langle\mathsf{a}\boxplus\rangle*\cdot\mathsf{h}\cdot\mathsf{x}\ \ \underline{\curlywedge}_{\mathsf{f}\cdot\mathsf{a}}\ \langle\mathsf{a}\boxtimes\rangle*\cdot\mathsf{h}\cdot\mathsf{y}
\end{array}
\qquad \text{(ii)}
$$

This looks like a downwards pass, in the sense that every element of the tree is replaced with some function of its ancestors, but it is not immediately obvious how it matches the pattern $\mathsf{g}*\circ\mathsf{paths}$. The following theorem makes the correspondence clear.

THEOREM (7) If h satisfies (ii) then h is a downwards pass. $\Diamond$

Consider the type daerht, pronounced 'dirt'; this is the least fixed point of the functor D sending X to $\mathsf{A} \mid (\mathsf{A} \parallel \mathsf{X}) \mid (\mathsf{A} \parallel \mathsf{X})$, and is given by the equation

$$\mathsf{daerht}\cdot\mathsf{A} \;=\; \diamond\cdot\mathsf{A} \;\mid\; \mathsf{A} \nearrow \mathsf{daerht}\cdot\mathsf{A} \;\mid\; \mathsf{A} \nwarrow \mathsf{daerht}\cdot\mathsf{A}$$

(We make no apology for using the same symbol $\diamond$ for singleton threads and for singleton daerhts.) Informally, daerhts are to threads as cons lists are to snoc lists. More formally, the correspondence between the two is given by the isomorphism

$$\mathsf{td} \;=\; (\!|\mathsf{H}\colon \diamond \curlyvee \boxslash \curlyvee \boxbslash|\!) \;\in\; \mathsf{thread}\cdot\mathsf{A} \to \mathsf{daerht}\cdot\mathsf{A}$$

where

$$\begin{aligned} \mathsf{p} \boxslash \mathsf{a} \;&=\; (\!|\mathsf{D}\colon \langle \nearrow \diamond\cdot\mathsf{a}\rangle \curlyvee \nearrow \curlyvee \nwarrow|\!)\cdot\mathsf{p} \\ \mathsf{p} \boxbslash \mathsf{a} \;&=\; (\!|\mathsf{D}\colon \langle \nwarrow \diamond\cdot\mathsf{a}\rangle \curlyvee \nearrow \curlyvee \nwarrow|\!)\cdot\mathsf{p} \end{aligned}$$

The operators $\boxslash$ and $\boxbslash$ effectively 'snoc' elements to daerhts.

It turns out that the function h of Theorem 7 satisfies

$$\mathsf{h} \;=\; (\!|\mathsf{D}\colon \mathsf{f} \curlyvee \boxplus \curlyvee \boxtimes|\!)* \circ \mathsf{td}* \circ \mathsf{paths}$$

To show this we will call upon the following lemma, concerning the operations $\oslash$ and $\obslash$ from the definition of paths.

LEMMA (8)

$$\begin{aligned} \mathsf{td} \circ \langle \mathsf{a}\oslash\rangle \;&=\; \langle \mathsf{a} \nearrow\rangle \circ \mathsf{td} \\ \mathsf{td} \circ \langle \mathsf{a}\obslash\rangle \;&=\; \langle \mathsf{a} \nwarrow\rangle \circ \mathsf{td} \end{aligned}$$

$\diamondsuit$

PROOF By Lemma 6,

$$\mathsf{td} \circ \langle \mathsf{a}\oslash\rangle \;=\; (\!|\langle \diamond\cdot\mathsf{a}\boxslash\rangle \curlyvee \boxslash \curlyvee \boxbslash|\!)$$

Also, $\nearrow$ associates with $\boxslash$—that is, $\langle \mathsf{a} \nearrow\rangle$ is $(\boxslash, \boxslash)$ F-promotable where $\mathsf{F}\cdot\mathsf{X} = \mathsf{X} \parallel \mathsf{A}$—because

$$\begin{aligned} & (\mathsf{a} \nearrow \mathsf{p}) \boxslash \mathsf{b} \\ =\;& (\!|\langle \nearrow \diamond\cdot\mathsf{b}\rangle \curlyvee \nearrow \curlyvee \nwarrow|\!)\cdot(\mathsf{a} \nearrow \mathsf{p}) \\ =\;& \qquad \{ \text{ catamorphisms } \} \\ & \mathsf{a} \nearrow ((\!|\langle \nearrow \diamond\cdot\mathsf{b}\rangle \curlyvee \nearrow \curlyvee \nwarrow|\!)\cdot\mathsf{p}) \\ =\;& \mathsf{a} \nearrow (\mathsf{p} \boxslash \mathsf{b}) \end{aligned}$$

Similarly, $\nearrow$ associates with $\boxbslash$, so $\langle \mathsf{a} \nearrow\rangle$ is $(\diamond \curlyvee \boxslash \curlyvee \boxbslash, (\langle \mathsf{a} \nearrow\rangle \circ \diamond) \curlyvee \boxslash \curlyvee \boxbslash)$ D-promotable, and

$$\langle \mathsf{a} \nearrow\rangle \circ \mathsf{td} \;=\; (\!|(\langle \mathsf{a} \nearrow\rangle \circ \diamond) \curlyvee \boxslash \curlyvee \boxbslash|\!)$$

Since $\langle \diamond\cdot\mathsf{a}\boxslash\rangle = \langle \mathsf{a} \nearrow\rangle \circ \diamond$, the first equation holds. The second is symmetric.

$\heartsuit$

We are now equipped to prove Theorem 7. Let $\circledast$ satisfy

$$\mathsf{u} \circledast_{\mathsf{a}} \mathsf{v} \;=\; \langle \mathsf{a}\boxplus\rangle{*}\cdot\mathsf{u} \;\mathbin{\underline{+}}_{\mathsf{f}\cdot\mathsf{a}}\; \langle \mathsf{a}\boxtimes\rangle{*}\cdot\mathsf{v}$$

so that $\mathsf{h} = \llparenthesis(\triangle\circ\mathsf{f})\curlyvee\circledast\rrparenthesis$. We will show that

$$\llparenthesis\mathsf{D}{:}\,\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis{*}\circ\mathsf{td}{*}\circ\mathsf{paths} \;=\; \llparenthesis(\triangle\circ\mathsf{f})\curlyvee\circledast\rrparenthesis$$

too.

Proof (of Theorem 7) On leaves we have

$$\llparenthesis\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis{*}\cdot\mathsf{td}{*}\cdot\mathsf{paths}\cdot\triangle\cdot\mathsf{a} \;=\; \triangle\cdot\mathsf{f}\cdot\mathsf{a}$$

while on branches we get

$$\begin{array}{ll}
 & \llparenthesis\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis{*}\cdot\mathsf{td}{*}\cdot\mathsf{paths}\cdot(\mathsf{x}\mathbin{\underline{+}}_{\mathsf{a}}\mathsf{y}) \\
= & \quad \{\ \mathsf{paths}\ \} \\
 & \llparenthesis\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis{*}\cdot\mathsf{td}{*}\cdot(\langle\mathsf{a}\oslash\rangle{*}\cdot\mathsf{paths}\cdot\mathsf{x} \;\mathbin{\underline{+}}_{\diamond\cdot\mathsf{a}}\; \langle\mathsf{a}\obslash\rangle{*}\cdot\mathsf{paths}\cdot\mathsf{y}) \\
= & \quad \{\ \text{Lemma 8}\ \} \\
 & \llparenthesis\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis{*}\cdot(\langle\mathsf{a}\nearrow\rangle{*}\cdot\mathsf{td}{*}\cdot\mathsf{paths}\cdot\mathsf{x} \;\mathbin{\underline{+}}_{\diamond\cdot\mathsf{a}}\; \langle\mathsf{a}\nwarrow\rangle{*}\cdot\mathsf{td}{*}\cdot\mathsf{paths}\cdot\mathsf{y}) \\
= & \quad \{\ \text{catamorphisms}\ \} \\
 & \langle\mathsf{a}\boxplus\rangle{*}\cdot\llparenthesis\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis{*}\cdot\mathsf{td}{*}\cdot\mathsf{paths}\cdot\mathsf{x} \;\mathbin{\underline{+}}_{\mathsf{f}\cdot\mathsf{a}}\; \langle\mathsf{a}\boxtimes\rangle{*}\cdot\llparenthesis\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis{*}\cdot\mathsf{td}{*}\cdot\mathsf{paths}\cdot\mathsf{y} \\
= & \quad \{\ \circledast\ \} \\
 & (\llparenthesis\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis{*}\cdot\mathsf{td}{*}\cdot\mathsf{paths}\cdot\mathsf{x}) \;\circledast_{\mathsf{a}}\; (\llparenthesis\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis{*}\cdot\mathsf{td}{*}\cdot\mathsf{paths}\cdot\mathsf{y})
\end{array}$$

Combining the two we get

$$\begin{array}{l}
\llparenthesis\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis{*}\circ\mathsf{td}{*}\circ\mathsf{paths}\circ(\triangle\curlyvee\mathbin{\underline{+}}) \\
\qquad = \;((\triangle\circ\mathsf{f})\curlyvee\circledast)\circ\mathsf{T}\cdot(\llparenthesis\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis{*}\circ\mathsf{td}{*}\circ\mathsf{paths})
\end{array}$$

and therefore

$$\llparenthesis\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis{*}\circ\mathsf{td}{*}\circ\mathsf{paths} \;=\; \llparenthesis(\triangle\circ\mathsf{f})\curlyvee\circledast\rrparenthesis \;=\; \mathsf{h}$$

♡

We have just seen that functions satisfying (ii) are downwards passes. Under what conditions are they accumulations? That is, under what conditions is $\llparenthesis\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis\circ\mathsf{td}$ a thread catamorphism? This question is answered next.

Definition (9) We say $(\mathsf{f},\boxplus,\boxtimes)$ *inverts to* $(\mathsf{f},\oplus,\otimes)$ if

$$\begin{array}{rcl}
\mathsf{b}\boxplus\mathsf{f}\cdot\mathsf{a} & = & \mathsf{f}\cdot\mathsf{b}\oplus\mathsf{a} \\
\mathsf{b}\boxtimes\mathsf{f}\cdot\mathsf{a} & = & \mathsf{f}\cdot\mathsf{b}\otimes\mathsf{a}
\end{array}$$

and $\boxplus$ and $\boxtimes$ each associate with both $\oplus$ and $\otimes$. We say $(\mathsf{f},\boxplus,\boxtimes)$ is *top down* if there exist $\oplus$ and $\otimes$ such that $(\mathsf{f},\boxplus,\boxtimes)$ inverts to $(\mathsf{f},\oplus,\otimes)$. ◇

Theorem (10) If $(\mathsf{f},\boxplus,\boxtimes)$ inverts to $(\mathsf{f},\oplus,\otimes)$ then

$$\llparenthesis\mathsf{D}{:}\,\mathsf{f}\curlyvee\boxplus\curlyvee\boxtimes\rrparenthesis\circ\mathsf{td} \;=\; \llparenthesis\mathsf{H}{:}\,\mathsf{f}\curlyvee\oplus\curlyvee\otimes\rrparenthesis$$

◇

Proof According to the promotion theorem, it is sufficient to show that $\llparenthesis \mathsf{f} \curlyvee \boxplus \curlyvee \boxtimes \rrparenthesis$ is $(\boxslash, \oplus)$ and $(\boxbslash, \otimes)$ promotable, that is, that the two equations

$$\begin{aligned} \llparenthesis \mathsf{f} \curlyvee \boxplus \curlyvee \boxtimes \rrparenthesis \cdot (\mathsf{p} \boxslash \mathsf{a}) &= \llparenthesis \mathsf{f} \curlyvee \boxplus \curlyvee \boxtimes \rrparenthesis \cdot \mathsf{p} \oplus \mathsf{a} \\ \llparenthesis \mathsf{f} \curlyvee \boxplus \curlyvee \boxtimes \rrparenthesis \cdot (\mathsf{p} \boxbslash \mathsf{a}) &= \llparenthesis \mathsf{f} \curlyvee \boxplus \curlyvee \boxtimes \rrparenthesis \cdot \mathsf{p} \otimes \mathsf{a} \end{aligned}$$

hold. We prove the first of these; the second is symmetric.

$$\begin{array}{ll}
 & \llparenthesis \mathsf{f} \curlyvee \boxplus \curlyvee \boxtimes \rrparenthesis \cdot (\mathsf{p} \boxslash \mathsf{a}) = \llparenthesis \mathsf{f} \curlyvee \boxplus \curlyvee \boxtimes \rrparenthesis \cdot \mathsf{p} \oplus \mathsf{a} \\
= & \qquad \{ \ \boxslash \ \} \\
 & \llparenthesis \mathsf{f} \curlyvee \boxplus \curlyvee \boxtimes \rrparenthesis \cdot \llparenthesis \langle \nearrow \diamond \cdot \mathsf{a} \rangle \curlyvee \nearrow \curlyvee \nwarrow \rrparenthesis \cdot \mathsf{p} = \llparenthesis \mathsf{f} \curlyvee \boxplus \curlyvee \boxtimes \rrparenthesis \cdot \mathsf{p} \oplus \mathsf{a} \\
= & \qquad \{ \text{ promotion, catamorphisms } \} \\
 & \llparenthesis \langle \boxplus \mathsf{f} \cdot \mathsf{a} \rangle \curlyvee \boxplus, \boxtimes \rrparenthesis \cdot \mathsf{p} = \llparenthesis \mathsf{f} \curlyvee \boxplus \curlyvee \boxtimes \rrparenthesis \cdot \mathsf{p} \oplus \mathsf{a} \\
\Leftarrow & \qquad \{ \text{ condition on } \boxplus \text{ and } \oplus \ \} \\
 & \mathsf{b} \boxplus \mathsf{f} \cdot \mathsf{a} = \mathsf{f} \cdot \mathsf{b} \oplus \mathsf{a} \ \wedge \ \llparenthesis (\langle \oplus \mathsf{a} \rangle \circ \mathsf{f}) \curlyvee \boxplus \curlyvee \boxtimes \rrparenthesis = \langle \oplus \mathsf{a} \rangle \circ \llparenthesis \mathsf{f} \curlyvee \boxplus \curlyvee \boxtimes \rrparenthesis \\
\Leftarrow & \qquad \{ \text{ promotion } \} \\
 & \mathsf{b} \boxplus \mathsf{f} \cdot \mathsf{a} = \mathsf{f} \cdot \mathsf{b} \oplus \mathsf{a} \ \wedge \ \langle \oplus \mathsf{a} \rangle \text{ is } (\boxplus, \boxplus) \text{ and } (\boxtimes, \boxtimes) \text{ promotable} \\
= & \qquad \{ \text{ definition } \} \\
 & \mathsf{b} \boxplus \mathsf{f} \cdot \mathsf{a} = \mathsf{f} \cdot \mathsf{b} \oplus \mathsf{a} \ \wedge \ \langle \oplus \mathsf{a} \rangle \text{ associates with } \boxplus \text{ and with } \boxtimes
\end{array}$$

$\heartsuit$

Corollary (11) If h satisfies (ii), and $(\mathsf{f}, \boxplus, \boxtimes)$ inverts to $(\mathsf{f}, \oplus, \otimes)$, then

$$\mathsf{h} = (\mathsf{f}, \oplus, \otimes) \Downarrow$$

$\diamondsuit$

For example, consider the function depths, which replaces every element of a tree with its depth in that tree; it satisfies

$$\begin{aligned} \mathsf{depths} \cdot \vartriangle \cdot \mathsf{a} &= \vartriangle \cdot 1 \\ \mathsf{depths} \cdot (\mathsf{x} \pm_{\mathsf{a}} \mathsf{y}) &= \langle 1+ \rangle * \cdot \mathsf{depths} \cdot \mathsf{x} \pm_1 \langle 1+ \rangle * \cdot \mathsf{depths} \cdot \mathsf{y} \end{aligned}$$

That is, depths satisfies (ii), with f being $!1$ and $\boxplus$ and $\boxtimes$ both being the $\odot$ such that $\mathsf{a} \odot \mathsf{u} = 1 + \mathsf{u}$; hence, by Theorem 7, depths is a downwards pass and a catamorphism. Computed naively, depths will take parallel time quadratic in the depth of the tree, but $(!1, \odot, \odot)$ inverts to $(!1, \tilde{\odot}, \tilde{\odot})$ and so

$$\mathsf{depths} = (!1, \tilde{\odot}, \tilde{\odot}) \Downarrow$$

and depths can be computed quickly too.

5 Parallel prefix

Accumulations on trees provide a valuable tool for abstraction, as we hope to show by the following example.

The *prefix sums* problem is a problem on non-empty lists, which are given by the equation

$$\mathsf{list}\cdot A \;=\; \square\cdot A \mid \mathsf{list}\cdot A \mathbin{+\!\!\!+} \mathsf{list}\cdot A$$

modulo the law that $\mathbin{+\!\!\!+}$ is associative. The list catamorphism $(\!|f \curlyvee \odot|\!)$ satisfies the equations

$$\begin{aligned}(\!|f \curlyvee \odot|\!)\cdot\square\cdot a &= f\cdot a\\ (\!|f \curlyvee \odot|\!)\cdot(x \mathbin{+\!\!\!+} y) &= (\!|f \curlyvee \odot|\!)\cdot x \odot (\!|f \curlyvee \odot|\!)\cdot y\end{aligned}$$

where the operator $\odot$ must also be associative. One example of a list catamorphism is the function last, which returns the last element of a list:

$$\mathsf{last} \;=\; (\!|\mathsf{id} \curlyvee \gg|\!)$$

Another is the function inits mentioned earlier:

$$\mathsf{inits} \;=\; (\!|(\square \circ \square) \curlyvee \oplus|\!) \qquad \text{where} \quad u \oplus v = u \mathbin{+\!\!\!+} \langle \mathsf{last}\cdot u \mathbin{+\!\!\!+}\rangle * \cdot v$$

The prefix sums problem is to evaluate the 'running totals' $(\!|f \curlyvee \odot|\!)* \circ \mathsf{inits}$ of a list. The operator $\odot$ must be associative; we also assume that it has a unit, e. For example, applied to the list $[a_1, \ldots, a_n]$, the problem is to compute

$$[f\cdot a_1,\; f\cdot a_1 \odot f\cdot a_2,\; \ldots,\; f\cdot a_1 \odot \cdots \odot f\cdot a_n]$$

This problem encapsulates a very common pattern of computation on lists; it has applications in, among other places, the evaluation of polynomials, compiler design, and numerous graph problems including minimum spanning tree and connected components (Akl, 1989).

It might appear from the above example that the problem inherently takes linear time to solve, even in parallel; the structure of the result seems to preclude any faster solution. However, Ladner and Fischer (1980), reworking earlier results by Kogge and Stone (1973) and Estrin (1960), show that the evaluation can be performed in logarithmic time on a linear number of processors acting in parallel. It turns out that their 'parallel prefix' algorithm, which we derive here, is naturally expressed in terms of accumulations on trees.

The problem is to evaluate

$$\mathsf{ps} \;=\; (\!|f \curlyvee \odot|\!)* \circ \mathsf{inits}$$

The first step in the derivation is to change the problem from one on lists to one on trees; the motivation for this is that trees often lead to logarithmic algorithms, whereas lists rarely do. So, we are looking for a quick function tps, which evaluates prefix sums on a tree, satisfying

$$\mathsf{fringe} \circ \mathsf{tps} \;=\; \mathsf{ps} \circ \mathsf{fringe} \qquad \text{(iii)}$$

where fringe is the function returning the leaves of a tree as a list:

$$\mathsf{fringe} \;=\; (\!|\square \curlyvee \oplus|\!) \qquad \text{where} \quad u \oplus_a v = u \mathbin{+\!\!\!+} v$$

We can calculate immediately the result of applying tps to a leaf, since

$$\mathsf{fringe}\cdot\mathsf{tps}\cdot\vartriangle\cdot\mathsf{a} \;=\; \square\cdot\mathsf{f}\cdot\mathsf{a}$$

and hence

$$\mathsf{tps}\circ\vartriangle \;=\; \vartriangle\circ\,\mathsf{f}$$

because fringe is injective on leaves. Letting $\mathsf{s} = (\!|\mathsf{f}\curlyvee\odot|\!)\circ\mathsf{fringe}$, we have on branches

$$\begin{array}{ll}
 & \mathsf{fringe}\cdot\mathsf{tps}\cdot(\mathsf{x}\pm_{\mathsf{a}}\mathsf{y}) \\
= & \qquad\left\{\text{ specification of } \mathsf{tps} \right\} \\
 & \mathsf{ps}\cdot\mathsf{fringe}\cdot(\mathsf{x}\pm_{\mathsf{a}}\mathsf{y}) \\
= & \qquad\left\{\ \mathsf{fringe}\ \right\} \\
 & \mathsf{ps}\cdot(\mathsf{fringe}\cdot\mathsf{x} \mathbin{+\!\!+} \mathsf{fringe}\cdot\mathsf{y}) \\
= & \qquad\left\{\ \mathsf{ps},\ \mathsf{inits}\ \right\} \\
 & (\!|\mathsf{f}\curlyvee\odot|\!)*\cdot(\mathsf{inits}\cdot\mathsf{fringe}\cdot\mathsf{x} \;\mathbin{+\!\!+}\; \langle\mathsf{fringe}\cdot\mathsf{x}\mathbin{+\!\!+}\rangle*\cdot\mathsf{inits}\cdot\mathsf{fringe}\cdot\mathsf{y}) \\
= & \qquad\left\{\ *,\ \text{catamorphisms}\ \right\} \\
 & (\!|\mathsf{f}\curlyvee\odot|\!)*\cdot\mathsf{inits}\cdot\mathsf{fringe}\cdot\mathsf{x} \;\mathbin{+\!\!+}\; \langle\mathsf{s}\cdot\mathsf{x}\odot\rangle*\cdot(\!|\mathsf{f}\curlyvee\odot|\!)*\cdot\mathsf{inits}\cdot\mathsf{fringe}\cdot\mathsf{y} \\
= & \qquad\left\{\ \mathsf{ps};\ \text{specification of } \mathsf{tps}\ \right\} \\
 & \mathsf{fringe}\cdot\mathsf{tps}\cdot\mathsf{x} \;\mathbin{+\!\!+}\; \langle\mathsf{s}\cdot\mathsf{x}\odot\rangle*\cdot\mathsf{fringe}\cdot\mathsf{tps}\cdot\mathsf{y}
\end{array}$$

This does not completely determine tps on branches, since fringe is not injective on branches, but it is 'sweetly reasonable' to suppose that

$$\mathsf{tps}\cdot(\mathsf{x}\pm_{\mathsf{a}}\mathsf{y}) \;=\; \mathsf{tps}\cdot\mathsf{x} \;\pm_{\mathsf{b}}\; \langle\mathsf{s}\cdot\mathsf{x}\odot\rangle*\cdot\mathsf{tps}\cdot\mathsf{y}$$

for some b; certainly, this supposition is consistent with the indirect specification (iii) of tps. The calculation can tell us nothing about b, the root of $\mathsf{tps}\cdot(\mathsf{x}\pm_{\mathsf{a}}\mathsf{y})$, because fringe throws branch labels away.

This gives us now a direct—that is, executable—specification of tps:

$$\begin{array}{rcl}
\mathsf{tps}\cdot\vartriangle\cdot\mathsf{a} & = & \vartriangle\cdot\mathsf{f}\cdot\mathsf{a} \\
\mathsf{tps}\cdot(\mathsf{x}\pm_{\mathsf{a}}\mathsf{y}) & = & \mathsf{tps}\cdot\mathsf{x} \;\pm_{\mathsf{b}}\; \langle\mathsf{s}\cdot\mathsf{x}\odot\rangle*\cdot\mathsf{tps}\cdot\mathsf{y}
\end{array} \tag{iv}$$

for some b. Executing this specification requires parallel time quadratic in the depth of the tree; we show next how to improve this to linear parallel time, by exploiting the freedom in the choice of value for b.

Suppose that $\mathsf{b} = \mathsf{s}\cdot\mathsf{x}$, that is, that

$$\mathsf{tps}\cdot(\mathsf{x}\pm_{\mathsf{a}}\mathsf{y}) \;=\; \mathsf{tps}\cdot\mathsf{x} \;\pm_{\mathsf{s}\cdot\mathsf{x}}\; \langle\mathsf{s}\cdot\mathsf{x}\odot\rangle*\cdot\mathsf{tps}\cdot\mathsf{y}$$

Intuitively, this allows the computation of $\mathsf{tps}\cdot(\mathsf{x} \pm_{\mathsf{a}} \mathsf{y})$ from $\mathsf{tps}\cdot\mathsf{x}$ and $\mathsf{tps}\cdot\mathsf{y}$ to be split into two parts, the first bringing $\mathsf{s}\cdot\mathsf{x}$ to the root of the tree and the second mapping $\langle \mathsf{s}\cdot\mathsf{x}\odot\rangle$ over the right child. More formally, suppose that

$$\begin{aligned} \mathsf{up}\cdot(\mathsf{x} \pm_{\mathsf{a}} \mathsf{y}) &= \mathsf{up}\cdot\mathsf{x} \pm_{\mathsf{s}\cdot\mathsf{x}} \mathsf{up}\cdot\mathsf{y} \\ \mathsf{down}\cdot(\mathsf{u} \pm_{\mathsf{b}} \mathsf{v}) &= \mathsf{down}\cdot\mathsf{u} \pm_{\mathsf{b}} \langle \mathsf{b}\odot\rangle{*}\cdot\mathsf{down}\cdot\mathsf{v} \end{aligned}$$

whence

$$\mathsf{down}\cdot\mathsf{up}\cdot(\mathsf{x} \pm_{\mathsf{a}} \mathsf{y}) = \mathsf{down}\cdot\mathsf{up}\cdot\mathsf{x} \pm_{\mathsf{s}\cdot\mathsf{x}} \langle \mathsf{s}\cdot\mathsf{x}\odot\rangle{*}\cdot\mathsf{down}\cdot\mathsf{up}\cdot\mathsf{y}$$

so $\mathsf{down}\circ\mathsf{up}$ follows the same pattern as tps. An inductive proof shows that

$$(\mathsf{down}\circ\mathsf{up} = \mathsf{tps}) \;\Leftarrow\; (\mathsf{down}\circ\mathsf{up}\circ\vartriangle = \mathsf{tps}\circ\vartriangle)$$

(We cannot use the unique extension property because we do not yet know whether tps is a catamorphism.) The premise of this implication is satisfied if

$$\begin{aligned} \mathsf{up}\cdot\vartriangle\cdot\mathsf{a} &= \vartriangle\cdot\mathsf{f}\cdot\mathsf{a} \\ \mathsf{down}\cdot\vartriangle\cdot\mathsf{b} &= \vartriangle\cdot\mathsf{b} \end{aligned}$$

We have not yet improved the efficiency; up and down both take parallel time quadratic in the depth of the tree. However, as the names suggest, up and down are upwards and downwards passes, and we know how to make such functions quick: we turn them into accumulations.

Let $\mathsf{sl} = \mathsf{root}\circ\mathsf{up}$, so

$$\begin{aligned} \mathsf{sl}\cdot\vartriangle\cdot\mathsf{a} &= \mathsf{f}\cdot\mathsf{a} \\ \mathsf{sl}\cdot(\mathsf{x} \pm_{\mathsf{a}} \mathsf{y}) &= \mathsf{s}\cdot\mathsf{x} \end{aligned}$$

Now,

$$\mathsf{up} = \mathsf{sl}{*}\circ\mathsf{subtrees}$$

so up is indeed an upwards pass. It is not an accumulation, because sl is not a catamorphism: $\mathsf{sl}\cdot(\mathsf{x} \pm_{\mathsf{a}} \mathsf{y})$ depends on $\mathsf{s}\cdot\mathsf{x}$ and not just on $\mathsf{sl}\cdot\mathsf{x}$. However, as this suggests, $\mathsf{s}\curlywedge\mathsf{sl}$ *is* a catamorphism,

$$\mathsf{s}\curlywedge\mathsf{sl} = \llbracket(\mathsf{f}\curlywedge\mathsf{f})\curlyvee((\odot\curlywedge\ll)\circ\ll^2)\rrbracket$$

(In fact, sl is a *zygomorphism* (Malcolm, 1990): a function which, although not catamorphic by itself, becomes catamorphic when tupled with another function—in this case, s—that is itself a catamorphism.) This means that

$$\begin{aligned} & \mathsf{up} \\ = \quad & \{ \text{ above } \} \\ & \mathsf{sl}{*}\circ\mathsf{subtrees} \end{aligned}$$

$$
\begin{array}{ll}
= & \quad \{ \text{ pairs } \} \\
& \gg\!* \circ (\mathsf{s} \curlywedge \mathsf{sl})* \circ \mathsf{subtrees} \\
= & \quad \{ \ \mathsf{s} \curlywedge \mathsf{sl} \text{ is a catamorphism } \} \\
& \gg\!* \circ \llbracket (\mathsf{f} \curlywedge \mathsf{f}) \curlyvee ((\odot \curlywedge \ll) \circ \ll^2) \rrbracket * \circ \mathsf{subtrees} \\
= & \quad \{ \ \Uparrow \ \} \\
& \gg\!* \circ (\mathsf{f} \curlywedge \mathsf{f}, (\odot \curlywedge \ll) \circ \ll^2) \Uparrow
\end{array}
$$

which can be evaluated quickly up to $\odot$.

So much for up; what about down? We have

$$
\begin{array}{rcl}
\mathsf{down} \cdot \triangle \cdot \mathsf{a} & = & \triangle \cdot \mathsf{a} \\
\mathsf{down} \cdot (\mathsf{u} \pm_{\mathsf{b}} \mathsf{v}) & = & \mathsf{down} \cdot \mathsf{u} \ \pm_{\mathsf{b}} \ \langle \mathsf{b} \odot \rangle * \cdot \mathsf{down} \cdot \mathsf{v}
\end{array}
$$

and so by Theorem 7 down is a downwards pass. Again, it is not an accumulation, because $(\mathsf{id}, \gg, \odot)$ is not top down. For, suppose $(\mathsf{id}, \gg, \odot)$ were to invert to $(\mathsf{id}, \oplus, \otimes)$; then by Theorem 10, the two functions

$$
\begin{array}{rcl}
\mathsf{f} & = & \llbracket \mathsf{H}\colon \mathsf{id} \curlyvee \gg \curlyvee \odot \rrbracket \\
\mathsf{g} & = & \llbracket \mathsf{D}\colon \mathsf{id} \curlyvee \oplus \curlyvee \otimes \rrbracket \circ \mathsf{td}
\end{array}
$$

would be equal. Consider now the three threads

$$
\begin{array}{rcl}
\mathsf{p} & = & \mathsf{a} \searrow \diamond \cdot \mathsf{b} \\
\mathsf{q} & = & \mathsf{a} \searrow (\mathsf{b} \swarrow \diamond \cdot \mathsf{c}) \\
\mathsf{r} & = & (\mathsf{a} \odot \mathsf{b}) \swarrow \diamond \cdot \mathsf{c}
\end{array}
$$

with a and c such that $\mathsf{a} \odot \mathsf{c}$ differs from c. We have

$$
\begin{array}{rclcrcl}
\mathsf{f} \cdot \mathsf{p} & = & \mathsf{a} \odot \mathsf{b} & \quad & \mathsf{g} \cdot \mathsf{p} & = & \mathsf{a} \otimes \mathsf{b} \\
\mathsf{f} \cdot \mathsf{q} & = & \mathsf{a} \odot \mathsf{c} & & \mathsf{g} \cdot \mathsf{q} & = & (\mathsf{a} \otimes \mathsf{b}) \oplus \mathsf{c} \\
\mathsf{f} \cdot \mathsf{r} & = & \mathsf{c} & & \mathsf{g} \cdot \mathsf{r} & = & (\mathsf{a} \odot \mathsf{b}) \oplus \mathsf{c}
\end{array}
$$

If f and g are to be equal, we see that $\otimes$ and $\odot$ must also be equal, in which case g returns the same values for q and r, whereas f returns different values.

So, down is not an accumulation. Consider, though, the fork of thread catamorphisms

$$
\llbracket !\mathsf{e} \curlyvee \gg \curlyvee \odot \rrbracket \curlywedge \llbracket \mathsf{id} \curlyvee \gg \curlyvee \odot \rrbracket
$$

This is itself a catamorphism (Fokkinga, 1990):

$$
\llbracket !\mathsf{e} \curlyvee \gg \curlyvee \odot \rrbracket \curlywedge \llbracket \mathsf{id} \curlyvee \gg \curlyvee \odot \rrbracket \ = \ \llbracket (!\mathsf{e} \curlywedge \mathsf{id}) \curlyvee \gg \curlyvee \boxtimes \rrbracket
$$

where

$$
\mathsf{a} \boxtimes (\mathsf{b}, \mathsf{c}) \ = \ (\mathsf{a} \odot \mathsf{b}, \mathsf{a} \odot \mathsf{c})
$$

Moreover, it is top down—it inverts to $\llbracket (!\mathsf{e} \curlywedge \mathsf{id}) \curlyvee \oplus \curlyvee \otimes \rrbracket$ where

$$
\begin{array}{rcl}
(\mathsf{b}, \mathsf{c}) \oplus \mathsf{d} & = & (\mathsf{b}, \mathsf{b} \odot \mathsf{d}) \\
(\mathsf{b}, \mathsf{c}) \otimes \mathsf{d} & = & (\mathsf{c}, \mathsf{c} \odot \mathsf{d})
\end{array}
$$

Thus,

$$\begin{array}{ll}
& \mathsf{down} \\
= & \quad \{ \text{ Theorem 7 } \} \\
& (\!|\mathsf{id} \curlyvee \gg \curlyvee \odot|\!)* \circ \mathsf{td}* \circ \mathsf{paths} \\
= & \quad \{ \text{ pairs } \} \\
& \gg * \circ (\!|(!\mathsf{e} \curlywedge \mathsf{id}) \curlyvee \gg \curlyvee \boxtimes|\!)* \circ \mathsf{td}* \circ \mathsf{paths} \\
= & \quad \{ \text{ Theorem 10 } \} \\
& \gg * \circ (!\mathsf{e} \curlywedge \mathsf{id}, \oplus, \otimes)\Downarrow
\end{array}$$

This gives us the promised efficient algorithm for tps, an upwards accumulation followed by a downwards accumulation, with maps after each accumulation to 'tidy up':

$$\mathsf{tps} \;=\; \gg * \circ (!\mathsf{e} \curlywedge \mathsf{id}, \oplus, \otimes)\Downarrow \circ \gg * \circ (\mathsf{f} \curlywedge \mathsf{f}, (\odot \curlywedge \ll) \circ \ll^2)\Uparrow$$

This is the essence of the parallel prefix algorithm.

6 Conclusion

We have presented two kinds of accumulation on binary trees: upwards accumulation, which captures the notion of passing information up from the leaves of a tree towards the root, and downwards accumulation, which corresponds to passing information in the other direction, from the root towards the leaves. We have given conditions under which the accumulations are both catamorphic and quick, that is, requiring parallel time proportional to the product of the depth of the tree and the time taken to perform the individual operations. We have shown how these accumulations neatly provide a solution to the prefix sums problem.

O'Donnell (1990) has presented a derivation similar to ours, without using accumulations; he only went as far as producing the characterization (iv) of tps, then developing a quick implementation of it as a single monolithic function without separating out the two phases. The result is a 'sweep' operation consisting of a tree of processes, each of which 'sends information in both directions on each data path'. Accumulations make the data flow much clearer, cleanly separating the two phases, and provide a systematic way of deriving an efficient solution to the problem.

The material presented here is covered in much greater depth in the author's thesis (Gibbons, 1991), but the interested reader is warned that the notation differs in places: different names have been used for the same concept, and even the same name for different concepts.

It turns out that the premise of Corollary 11, under which a homomorphic downwards pass can be computed quickly, is sufficient to allow the accumulation to be computed on a CREW PRAM machine in parallel time proportional to the product of the logarithm of the depth of the tree and the time taken by the individual operations (Gibbons, 1992). It also appears that upwards accumulation can be computed on an EREW PRAM in parallel time proportional to the product of the depth of the tree and the time taken by the individual operations (Abrahamson et al., 1989). In view of these facts, upwards and downwards accumulations might form suitable primitives for a data parallel programming language.

Another topic to explore is the generalization of accumulations to arbitrary initial data types. Meertens (1990) gives a general construction yielding the subtrees of a tree, and upwards accumulations are a special case of his *paramorphisms*, but it is by no means obvious how to generalize downwards accumulations to other data types.

The author is grateful to Richard Bird, Geraint Jones, Lambert Meertens, David Skillicorn, and the anonymous referees, all of whom have made suggestions which have improved the presentation of this paper.

References

K. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka (1989). *A simple parallel tree contraction algorithm.* Journal of Algorithms, 10:287–302.

Selim G. Akl (1989). *Design and Analysis of Parallel Algorithms.* Prentice-Hall.

Roland Backhouse (1989). *An exploration of the Bird-Meertens formalism.* In *International Summer School on Constructive Algorithmics, Hollum, Ameland.* STOP project. Also available as Technical Report CS 8810, Department of Computer Science, Groningen University, 1988.

Richard S. Bird (1987). *An introduction to the theory of lists.* In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*, pages 3–42. Springer-Verlag. Also available as Technical Monograph PRG-56, from the Programming Research Group, Oxford University.

Richard S. Bird (1988). *Lectures on constructive functional programming.* In Manfred Broy, editor, *Constructive Methods in Computer Science.* Springer-Verlag. Also available as Technical Monograph PRG-69, from the Programming Research Group, Oxford University.

G. Estrin (1960). *Organization of computer systems—the fixed plus variable structure computer.* In *Proceedings Western Joint Computer Conference,* pages 33–40.

Maarten M. Fokkinga (1990). *Tupling and mutumorphisms.* The Squiggolist, 1(4):81–82.

Jeremy Gibbons (1991). *Algebras for Tree Algorithms.* D. Phil. thesis, Programming Research Group, Oxford University. Available as Technical Monograph PRG-94.

Jeremy Gibbons (1992). *Fast downwards accumulations on trees.* Department of Computer Science, University of Auckland. Submitted to 16th Australian Computer Science Conference.

Tatsuya Hagino (1987). *A Categorical Programming Language.* PhD thesis, Laboratory for the Foundations of Computer Science, Edinburgh.

Peter M. Kogge and Harold S. Stone (1973). *A parallel algorithm for the efficient solution of a general class of recurrence equations.* IEEE Transactions on Computers, C-22(8):786–793.

Richard E. Ladner and Michael J. Fischer (1980). *Parallel prefix computation.* Journal of the ACM, 27(4):831–838.

Grant Malcolm (1990). *Algebraic Data Types and Program Transformation.* PhD thesis, Rijksuniversiteit Groningen.

Lambert Meertens (1986). *Algorithmics: Towards programming as a mathematical activity.* In J. W. de Bakker, M. Hazewinkel, and J. K. Lenstra, editors, *Proc. CWI Symposium on Mathematics and Computer Science*, pages 289–334. North-Holland.

Lambert Meertens (1990). *Paramorphisms.* Technical Report CS-R9005, CWI, Amsterdam.

John T. O'Donnell (1990). *Derivation of fine-grain algorithms.* Presentation at IFIP Working Group 2.8 meeting, Rome.