LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - The University of Edinburgh

Tail recursion via universal invariants

by

C. Barry Jay

Tail recursion via universal invariants

LFCS Report Series

ECS-LFCS-91-151

April 1991

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

Tail recursion via universal invariants

C. Barry Jay *
University of Edinburgh
University of Ottawa

April 23, 1991

Abstract

The iteration step of a recursion can be modelled by a loop diagram in a category, with the result of the recursion given by the colimit of the loop. For tail recursion on lists this yields the familiar foldleft operation, whose properties follow from the universal property of the colimit. By contrast, the initial algebra approach to lists yields foldright, whose implementation is often the less efficient of the two. Under mild conditions, however, the two definitions of lists are equivalent.

*Research supported by The Royal Society of Edinburgh/BP, and NSERC operating grant OGPIN 016.

1 Introduction

The domain equation for lists on an object A

$$L \cong 1 + (A \times L)$$

may have many solutions, which can be thought of as *stacks* (see below). Uniqueness of the object of finite lists is usually forced by requiring an initial algebra for the corresponding functor $F(-) = 1 + (A \times -)$, i.e. an *initial stack*, while for power we must further assume that it is a *parametrised* (initial) stack (which follows from initiality in the presence of, say, cartesian closure). They have pleasing mathematical properties, but give a poor account of lists as actually used in computation. The reason for this is quite simple. If L is a parametrised stack on A and C has a left A-action, i.e. a morphism $\alpha' : A \times C \rightarrow C$, then initiality yields the operation foldright $\texttt{foldr}(\alpha') : L \times C \rightarrow C$ whose action is typically given by

$$\texttt{foldr}(\alpha')([a_0, a_1, \ldots, a_n], x) = a_0 * (a_1 * (\ldots (a_n * x) \ldots))$$

where $a * x = \alpha'(a, x)$. An implemention of this operation has two major drawbacks: the list must be deconstructed to obtain its last entry before any computation can begin, and; it is messy if any of the entries are ill-defined.

Computationally more natural is tail recursion, whose basic operation is foldleft $\texttt{foldl}(\alpha) : C \times L \rightarrow C$ (where α is a right A-action on C, say, $\alpha = \alpha' \circ c$ where c is the symmetry for the product) given by

$$\texttt{foldl}(\alpha)(x, [a_0, a_1, \ldots, a_n]) = (\ldots ((x * a_0) * a_1) \ldots)$$

(where $*$ denotes the action). This operation avoids the defects of foldright mentioned above, but till now it has played a derivative part in the categorical semantics, being defined as, say,

$$\texttt{foldl}(\alpha) = \texttt{foldr}(\alpha') \circ (\texttt{rev} \times 1) \circ c$$

where $\texttt{rev}$ is the reverse operation. Conversely, foldright can be defined as

$$\texttt{foldr}(\alpha') = \texttt{foldl}(\alpha) \circ (\texttt{rev} \times 1) \circ c$$

These translations are neither efficient nor illuminating, however, so that tail recursive definitions typically proceed in an ad hoc fashion, only exploiting the universal property of the initial algebra indirectly. The chief conceptual advance of this paper is to marry standard practice to a clean semantics, by representing foldleft as the colimit, or universal invariant, of a loop.

A program loop can be represented by a categorical diagram which has exactly one object and one morphism (hence an endomorphism). The limit of such a loop diagram $f : C \rightarrow C$ is its fix object $\texttt{fix}(f) : \texttt{Fix}(f) \rightarrow C$. An invariant for the loop is a morphism $g : C \rightarrow Q$ whose value is unchanged by prior applications of the loop, i.e. $g \circ f = f$. Such invariants are cocones for the diagram. The result of iterating the loop is given by the colimit of the diagram, i.e. its *object of invariants* denoted $\texttt{inv}(f) : C \rightarrow Inv(f)$. A desirable property for a loop is that it always terminates or, a little more generally, that it always converges to a fixpoint. When this occurs then $\texttt{Inv}(f) \cong \texttt{Fix}(f)$ and the loop is said to *converge*. When applied to tail recursion we arrive at the notion of *convergent stacks*. Thus, unlike arbitrary limits and colimits, (e.g. equalisers and coequalisers) those of loops have a clear semantic significance.

The main results of this paper give conditions under which parametrised stacks are equivalent to convergent stacks. The context for these results is a bicartesian category, perhaps with equalisers, in which the products distribute over the sums. In the presence of a parametrised natural numbers object the correspondence extends to yet a third notion of list object, where the tail morphism is not merely required to converge, but to terminate (become stationary after a finite number of steps). The correspondence between initial stacks and these *terminating stacks* is Cockett's main subject matter in [2]. The introduction here of convergence simplifies the proofs while simultaneously weakening the hypotheses. Thus, the additional structure used there (a strong form of distributivity) is only required for proving additional desirable features of list constructions, such as showing that the list functor preserves connected limits, or that the list construction is stable under slicing. These results can be used to give a semantics for vectors, matrices, inner products and matrix multiplication, etc. [8].

In the other direction, one may wish to weaken the hypotheses to consider products and sums which are not cartesian, but perhaps derived from cartesian products. For example, the category of bottomless c.p.o's $\mathbf{Pos}(\omega)$ is bicartesian and satisfies the distributive law. Its sub-category of c.p.o's with bottom and lazy maps $\mathbf{Pos}(\omega)_\perp$ has cartesian product $\otimes$ and separated sum $\oplus$ (not cartesian) for which distributivity fails. Now the lazy list object on A is the parametrised stack for the functor $FX = 1\oplus(A\otimes X)$. It is also the convergent stack, in the appropriate sense. Note, however, that now $\mathtt{rev} \circ \mathtt{rev}$ is only the identity on total lists, while partial lists are mapped to $\perp$, and so serves as a test for totality. Thus, the two forms of folding are no longer interchangeable.

Yet again, we may consider partial maps in $\mathbf{Pos}(\omega)$ or equivalently, c.p.o's with $\perp$ and strict maps. Passing from total to partial maps leaves the parametrised, and convergent stacks unchanged [8], provided the notions of limit and colimit are interpreted in the appropriate, order-theoretic, sense [6].

Finally, when considering more general datatypes, such as labelled trees, gulf between the two approaches widens even further. The initial algebra approach begins computing at the tips and is useful for evaluating terms etc. The loopy approach, however, begins at the root, and is correct for pattern matching, etc. These differences are familiar from search algorithms, where the corresponding approaches are *depth-first* versus *breadth-first* search. A fuller treatment of trees must await further work.

2 Loops

Given a category $\mathcal{C}$ define $\mathcal{C}^c$ to be the category of loop diagrams in $\mathcal{C}$. Its objects are the endomorphisms $f : C \rightarrow C$ of $\mathcal{C}$ thought of as diagrams with *one* object and one arrow. The *loop morphisms* from f to $g : D \to D$ are the usual diagram morphisms,

$$\begin{tikzcd} C \arrow[loop left, "f"] \arrow[r, "h"] & D \arrow[loop right, "g"] \end{tikzcd}$$

i.e. $h : C \to D$ such that $h \circ f = g \circ h : C \to D$. Note that $\mathcal{C}^c$ is *not* a full subcategory of $\mathcal{C}^{\to}$ the arrow category of $\mathcal{C}$.

Loops may be used to represent the conditional of a while-loop or the unfolding of a fixpoint combinator [5]. We shall see that their limits and colimits play an important semantic role, even if they do not explicitly represent datatypes in a programming language. The semantic significance of other colimits, such as co-equalisers, is not so clear. It may be valuable to consider a semantic category which is bicartesian and has colimits of loops, without requiring full cocompleteness. Certainly loops deserve a closer treatment than heretofore.

The (categorical) limit of such a loop f on C, if it exists, is its *fix object* denoted $\mathtt{Fix}(f)$ with inclusion $\mathtt{fix}(f) : \mathtt{Fix}(f) \to C$. Thus any morphism $x : B \to C$ which is fixed by f (that is $f \circ x = x$) factors through $\mathtt{fix}(f)$.

An *invariant* of the loop f above is a morphism $g : C \to Q$ such that $g \circ f = g$, that is, a *cocone* for the loop diagram [13]). The colimit of the loop, if it exists, is its *universal invariant*, denoted $\mathtt{inv}(f) : C \to \mathtt{Inv}(f)$ whose codomain is its *object of invariants*. Then the invariant g factors through $\mathtt{inv}(f)$ in a unique way. Of course the object of invariants is only determined up to isomorphism. When f interprets the conditional of a while-loop on c.p.o.'s then $\mathtt{Inv}(f)$ is a sum of $\mathtt{Fix}(f)$ and an object of infinite loops [5].

Definition 2.1 *The loop f on C in $\mathcal{C}$* converges *if it has both a fix object and an object of invariants such that $Inv(f) = \mathtt{Fix}(f)$ and $\mathtt{inv}(f) \circ \mathtt{fix}(f) = 1 : Fix(f) \to Fix(f)$.*

$$\begin{tikzcd} \mathtt{Fix}(f) \arrow[r, "\mathtt{fix}(f)"'] & C \arrow[loop above, "f"] \arrow[r, "\mathtt{inv}(f)"'] & \mathtt{Fix}(f) \end{tikzcd}$$

In **Sets**, for example, there are no limiting processes so that convergence implies that every element of C is mapped to a fixpoint of f after a finite (but unknown) number of iterations. With domains it may happen that iteration of f never terminates but yet the sequence of iterations converges to a limiting fixpoint, e.g. f is increasing.

The mere existence of the colimit $\mathtt{inv}(f)$ of the loop does not provide much computational power; that it is a split epimorphism allows us to compute the comparison map for an invariant $g : C \to Q$ as $g \circ \mathtt{fix}(f) : \mathtt{Fix}(f) \to Q$. As a trivial application we have the following

Lemma 2.2 *If $f : C \to C$ converges and $g : C \to \mathtt{Fix}(f)$ is an invariant for which $g \circ \mathtt{fix}(f) = 1$ then $g = \mathtt{inv}(f)$.* □

3 Stacks

Let $\mathcal{C}$ be a bicartesian category, i.e. have all finite cartesian products and sums. The pairing of $f : C \rightarrow A$ and $g : C \rightarrow B$ is denoted $\langle f, g \rangle : C \rightarrow A \times B$ with projections $\pi_{A,B} : A \times B \rightarrow A$ and $\pi'_{A,B} : A \times B \rightarrow B$, symmetry $c_{A,B} : A \times B \rightarrow B \times A$ and diagonal $\delta = \langle \mathtt{id}, \mathtt{id} \rangle$. The unit morphisms are $l : 1 \times A \rightarrow A$ and $r : A \times 1 \rightarrow A$. The terminal object is 1 (though $1 : A \rightarrow A$ may also denote the identity morphism on A and the successor of 0, according to context) with $!_A : A \rightarrow 1$ the corresponding morphism. Given a constant $b : 1 \rightarrow B$ then $b \circ ! : A \rightarrow B$ may be abbreviated by $b : A \rightarrow B$. Usually, the bracketing of products and applications of associativity will be suppressed, as will the subscripts on natural transformations, unless they aid comprehension.

The cases morphism for $f : A \rightarrow C$ and $g : B \rightarrow C$ is $[f, g] : A + B \rightarrow C$ with inclusions $\iota : A \rightarrow A + B$ and $\iota' : B \rightarrow A + B$.

If, for each triplet of objects A, B, C the morphism

$$(A \times B) + (A \times C) \xrightarrow{[1 \times \iota, 1 \times \iota']} A \times (B + C)$$

is an isomorphism (with inverse denoted $d_{A,B,C}$) then $\mathcal{C}$ satisfies the *distributive law* (for products over sums) and is called a *polynomial* category, since, up to isomorphism, polynomial functors satisfy the usual identities. Elsewhere they have been called *distributive* categories [18], a title now reserved for a stronger condition, or *pre-distributive* categories with intial object [3] which is too diminutive for such a fundamental concept. Examples include all bicartesian closed categories (since $A \times (-)$ is a left adjoint) but also many non-closed categories such as the category of topological spaces and continuous functions, and many of its sub-categories, e.g. of metric spaces.

Let A be an object in $\mathcal{C}$. A *stack* on A [18] is given by an object S together with morphisms $\mathtt{empty} : 1 \rightarrow S$ and $\mathtt{push} : A \times S \rightarrow S$ such that we have the following isomorphism

$$S \xleftarrow[\simeq]{[\mathtt{empty}, \mathtt{push}]} 1 + (A \times S)$$

The inverse is called $\mathtt{pop} : S \rightarrow 1 + (A \times S)$. If the intention is that pop is only to be applied to non-empty stacks then a result of empty can be interpreted as failure. A stack may consist of finite or infinite lists, etc. When the stack is to be thought of as a list object L then we may rename empty as nil and push as cons. Here are some of the typical list operations

$$\begin{aligned} \mathtt{head} &= (1 + \pi) \circ \mathtt{pop} : L \rightarrow 1 + A \\ \mathtt{tail} &= [\mathtt{nil}, \pi'] \circ \mathtt{pop} : L \rightarrow L \\ \eta &= [\cdot] = \mathtt{cons} \circ \langle 1, \mathtt{nil} \rangle : A \rightarrow A \times L \rightarrow L \end{aligned}$$

Note that, in general, stacks cannot be appended.

The iteration step of tail recursion can be described by a loop as follows. Let $(L, \mathtt{nil}, \mathtt{cons})$ be a stack for A and consider a right A-action $\alpha : C \times A \rightarrow C$ on an object C. Define the loop $\mathtt{shunt}(\alpha) : C \times L \rightarrow C \times L$ by

$$C \times L \xrightarrow{d \circ (1 \times \mathtt{pop})} (C \times 1) + (C \times A \times L) \xrightarrow{[1 \times \mathtt{empty}, \alpha \times 1]} C \times L$$

When $C = 1$ with its unique A-action then `shunt` corresponds to `tail`.

$\mathtt{shunt}(\alpha)$ makes the head of the stack (if it has one) act on C and yields the result of the action and the tail as its value. If L consists of finite lists then iterating $\mathtt{shunt}(\alpha)$ will ultimately yield values whose stack component is `nil`, i.e. which are fixed by $\mathtt{shunt}(\alpha)$. Thus $\mathtt{shunt}(\alpha)$ is then convergent.

Fix the polynomial category $\mathcal{C}$ as the ambient category for the rest of the paper, together with a stack $\mathbf{L} = (L, \mathtt{nil}, \mathtt{cons})$ on some object A. Further, let $B, C, \ldots$ denote objects of $\mathcal{C}$ and $x : B \to C$, $f : C \to C$, $\alpha : C \times A \to C$ and $\alpha' = \alpha \circ c : A \times C \to C$ denote some of its morphisms, not necessarily fixed.

4 Convergent Stacks

$\mathbf{L}$ is a *convergent stack* if for all objects C with action α the loop $\mathtt{shunt}(\alpha)$ converges with fixpoints given by $\langle 1, \mathtt{nil}\rangle : C \to C \times L$. The corresponding universal invariant is *foldleft* of α denoted $\mathtt{foldl}(\alpha) : C \times L \to C$.

In **Sets** and $\mathbf{Pos}(\omega)$ this is the usual operation *foldleft* on lists. Note that invariants for $\mathtt{shunt}(\alpha)$ are co-equalising for the pair

$$C \times A \times L \underset{1 \times \mathtt{cons}}{\overset{\alpha \times 1}{\rightrightarrows}} C \times L$$

whose co-equaliser is just $\mathtt{foldl}(\alpha)$. While this description is useful for analytic purposes, the special nature of the loop, in contrast to general coequalisers, would be lost if this were taken as the definition.

The following lemma demonstrates a basic proof technique.

Lemma 4.1 *If $\beta : D \times A \to D$ is another right A-action and $h : C \to D$ is an A-action homomorphism then the following diagram commutes.*

$$\begin{tikzcd} C \times L \arrow[r, "\mathtt{foldl}(\alpha)"] \arrow[d, "h \times 1"'] & C \arrow[d, "h"] \\ D \times L \arrow[r, "\mathtt{foldl}(\beta)"'] & D \end{tikzcd}$$

Proof

$$\begin{aligned} \mathtt{foldl}(\beta) \circ (h \times 1) \circ (\alpha \times 1) &= \mathtt{foldl}(\beta) \circ (\beta \times 1) \circ (h \times 1) \\ &= \mathtt{foldl}(\beta) \circ (1 \times \mathit{cons}) \circ (h \times 1) \\ &= \mathtt{foldl}(\beta) \circ (h \times 1) \circ (1 \times \mathtt{cons}) \end{aligned}$$

Hence $\mathtt{foldl}(\beta) \circ (h \times 1)$ is an invariant for $\mathtt{shunt}(\alpha)$ and so equals $g \circ \mathtt{foldl}(\alpha)$ for some $g : C \to D$ which we can now prove is h:

$$\begin{aligned} g &= g \circ \mathtt{foldl}(\alpha) \circ \langle 1, \mathtt{nil}\rangle \\ &= \mathtt{foldl}(\beta) \circ (h \times 1) \circ \langle 1, \mathtt{nil}\rangle \\ &= \mathtt{foldl}(\beta) \circ \langle 1, \mathtt{nil}\rangle \circ h \\ &= h \end{aligned}$$

$\square$

Define

$$
\begin{aligned}
\texttt{rev} &= \texttt{foldl}(\mathsf{cons} \circ c) \circ \langle \texttt{nil}, 1\rangle : L \to L^2 \to L \\
\texttt{snoc} &= \texttt{rev} \circ \mathsf{cons} \circ (1 \times \texttt{rev}) \circ c : L \times A \to L \\
@ &= \texttt{foldl}(\texttt{snoc}) : L^2 \to L
\end{aligned}
$$

`rev` reverses a list, snoc is like cons except that it attaches the new entry to the tail of the list, rather than the head, and @ is the append function. Here are three lemmas which illustrate some of the proof techniques available.

Lemma 4.2 *The following diagram commutes (for $\alpha = \mathsf{cons} \circ c : L \times A \to L$).*

$$
\begin{tikzcd}
L^2 \arrow[r, "\texttt{foldl}(\alpha)"] \arrow[d, "c"'] & L \arrow[d, "\texttt{rev}"] \\
L^2 \arrow[r, "\texttt{foldl}(\alpha)"'] & L
\end{tikzcd}
$$

Thus $\texttt{rev} \circ \texttt{rev} = \mathsf{id}$.

Proof $\texttt{foldl}(\alpha) \circ c$ is an invariant for the loop $\mathsf{shunt}(\alpha)$ on L^2. Composing with $\langle 1, \texttt{nil}\rangle$ yields the commutativity of the square above. Stacking two of these squares together yields the second result. $\square$

Lemma 4.3 (i) $@ \circ \langle 1, \texttt{nil}\rangle = 1$

(ii) $@ \circ (1 \times \eta) = \mathsf{snoc}$

(iii) $@ \circ (\texttt{rev} \times \texttt{rev}) \circ c = \texttt{rev} \circ @$

(iv) $@ \circ \langle \texttt{nil}, 1\rangle = 1$

(v) $@ \circ (\eta \times 1) = \mathsf{cons}$

(vi) @ *is associative.*

(vii) $\mathsf{snoc} \circ (\mathsf{cons} \times 1) = \mathsf{cons} \circ (1 \times \mathsf{snoc})$

Proof (i): by definition. (ii):

$$
\begin{aligned}
@ \circ (1 \times \eta) &= @ \circ (1 \times \mathsf{cons} \circ \langle 1, \texttt{nil}\rangle) \\
&= @ \circ \langle \mathsf{snoc}, \texttt{nil}\rangle \\
&= @ \circ \langle 1, \texttt{nil}\rangle \circ \mathsf{snoc} = \mathsf{snoc}
\end{aligned}
$$

(iii): Use (i) and (ii) to show that the lefthand side is an invariant for $\mathtt{shunt(snoc)}$.
(iv): This follows from (i),(iii) and Lemma 0.4.2.

$$
\begin{aligned}
@ \circ \langle \mathtt{nil}, 1\rangle &= \mathtt{rev} \circ \mathtt{rev} \circ @ \circ \langle \mathtt{nil}, 1\rangle \\
&= \mathtt{rev} \circ @ \circ (\mathtt{rev}\times\mathtt{rev}) \circ c \circ \langle \mathtt{nil}, 1\rangle \\
&= \mathtt{rev} \circ @ \circ (\mathtt{rev}\times\mathtt{rev})\langle 1, \mathtt{nil}\rangle \\
&= \mathtt{rev} \circ @ \circ \langle 1, \mathtt{nil}\rangle \circ \mathtt{rev} \\
&= \mathtt{rev} \circ \mathtt{rev} = 1
\end{aligned}
$$

(v): Follows from (iv).
(vi): It suffices to prove that $@ \circ (@\times 1) : L^3 \to L$ and $@ \circ (1\times @)$ are both the colimit of the diagram with one object L^3 and two loops which are given by $\mathtt{shunt(snoc)}\times 1$ and $1\times\mathtt{shunt(snoc)}$. Their equality follows since they have a common splitting, namely $1\times\mathtt{nil}\times\mathtt{nil}$ (c.f. Lemma 0.2.2). Each of these loops is a shunt morphism (up to loop isomorphism), and so converges. Now Lemma 0.2.2 shows that their universal invariants are given by $@\times 1$ and $1\times @$ respectively. Hence if $f : L^3 \to Q$ is invariant for both loops then it factors through each of them and, since $1\times\mathtt{nil}\times 1$ is a splitting for both colimits, by the same map $f' = f \circ (1\times\mathtt{nil}\times 1)$. Now

$$
f' \circ (\mathtt{snoc}\times 1) = f' \circ (@\times 1) \circ (1\times\eta\times 1) = f' \circ (1\times @) \circ (1\times\eta\times 1) = f' \circ (1\times\mathtt{cons})
$$

shows that f' is an invariant for $\mathtt{shunt(snoc)}$ and so f' factors through $@$ as required. The factorisation is unique since $@ \circ (@\times 1)$ is a split epi.
(vii): Compose the associativity equation with $\eta\times 1\times\eta : A\times L\times A \to L^3$. □

5 Parametrised Stacks

The traditional method of specifying finite lists is to require of $\mathbf{L}$ that $(L, [\mathtt{nil}, \mathtt{cons}])$ is an an initial algebra for the functor $F : \mathcal{C} \to \mathcal{C}$ given by $F(-) = 1 + (A\times -)$. Of course initiality forces $\mathbf{L}$ to be a stack, namely the *initial stack* [10, 11, 17].

An initial stack L on A is *parametrised* if for each object B the functor $G(-) = B + (A\times -)$ has initial algebra $L\times B$. More precisely, each x and α' have a corresponding *foldright* morphism $\mathtt{foldr}(x, \alpha') : L\times B \to C$ which is the unique morphism making the following diagram commute

$$
\begin{tikzcd}
B \arrow[r, "\langle \mathtt{nil}, 1\rangle"] \arrow[dr, "x"'] & L\times B \arrow[d, dashed, "\mathtt{foldr}(x,\alpha)"] & A\times L\times B \arrow[l, "\mathtt{cons}\times 1"'] \arrow[d, dashed, "1\times\mathtt{foldr}(x,\alpha)"] \\
& C & A\times C \arrow[l, "\alpha"]
\end{tikzcd}
$$

If $x = 1_C$ then it is denoted by $\mathtt{foldr}(\alpha')$. In **Sets** this is the usual operation *foldright* on lists. For example, appending of lists is given by $@' = \mathtt{foldr}(\mathtt{cons}) : L\times L \to L$. Let us pause briefly to compare this with our previous definition of append.

Lemma 5.1 *If L is both a parametrised and convergent stack for A $@ = @'$.*
Proof We will show that $@ = \mathtt{foldr(cons)}$. Lemma 0.4.3 (i) yields the `nil` step. For cons show that $@ \circ \mathtt{cons}$ is an invariant for $1\times\mathtt{shunt(snoc)}$ and so factors through its universal invariant $1\times @$ by some morphism, which proves to be cons. $\square$

Of course, if S is any stack on A then $R = S\times B$ satisfies the domain equation $R \cong GR$ by distributivity, so that stacks are automatically parametrised.

Lemma 5.2 *If C is cartesian closed then initial stacks are parametrised.*
Proof Given $f : A\times B \to C$ and $k : A \to B \to C$ then $\check{f} : A \to B \to C$ and $\hat{k} : A\times B \to C$ denote their transposes under the product-hom adjunction, and $e : (B \to C)\times B \to C$ denotes evaluation. Then define $\mathtt{foldr}(x, \alpha')$ to be the transpose of

$$L \xrightarrow{\mathtt{foldr}((x \circ l)\check{}, (\alpha' \circ (1\times e))\check{})} B \to C$$

$\square$

Assume that **L** is a parametrised stack for A for the rest of this section.

Note that it is usually much simpler to construct the parametrised forms directly than to fiddle about with transposition, as can be seen from constructing $@'$.

A (parametric) stack on 1 is a *(parametric) natural numbers object* (or NNO) typically denoted $(N, 0, S)$ where $0 = \mathtt{nil} : 1 \to N$ and successor $S = \mathtt{cons} \circ l^{-1} : N \to 1\times N \to N$ (see, e.g. [12, 16, 15, 4]). Its parametric universal property is then

$$\begin{tikzcd}
B \arrow[r, "{(0,\mathrm{id})}"] \arrow[dr, "x"'] & N\times B \arrow[d, dashed, "{\mathtt{It}(x,f)}"] & N\times B \arrow[l, "S\times\mathrm{id}"'] \arrow[d, dashed, "{\mathtt{It}(x,f)}"] \\
& C & C \arrow[l, "f"]
\end{tikzcd}$$

where $\mathtt{It}(x, f)$ is the *iterator* of x and f. Addition on natural numbers is given by $+ = \mathtt{It}(S) : N\times N \to N$ which is easily proved associative, unitary and commutative.

In the presence of a NNO L has an associated *length* morphism

$$\# = \mathtt{foldr}(0, S \circ \pi') \circ r^{-1} : L \to L\times 1 \to N$$

whence

$$\mathtt{foldr}(x, f \circ \pi') = \mathtt{It}(x, f) \circ (\#\times 1) : L\times C \to C$$

Thus, even in the absence of a NNO, we may define the lefthand side above to be the *A-iterator* of x and f denoted $\mathtt{It}_A(x, f)$. Thus, any parametrised stack may stand for the natural numbers. (Note, however, that the successor operation can only be represented if their is, say, a global element $1 \to A$.) Subscripts on `It` will be dropped when no loss of clarity will result.

Lemma 5.3 *Given f and x as above then*

(i) $f \circ \mathtt{It}(f) = \mathtt{It}(f) \circ (1\times f) : L\times C \to C$.

(ii) $\mathtt{It}(f)\circ(\mathtt{cons}\times 1) = \mathtt{It}(f)\circ(\pi'\times f) : (A\times L)\times C \to C$

(iii) *If* $f\circ x = x$ *then* $\mathtt{It}(f)\circ(1\times x) = x\circ\pi' : L\times B\to C$.

(iv) *If* $g\circ f = g$ *then* $g\circ\mathtt{It}(x,f) = g\circ x\circ\pi'$.

Proof Both sides equal: (i)$\mathtt{It}(f,f)$; (ii) $\mathtt{It}(f\circ\pi'$; (iii) $\mathtt{It}(x,f)$ and; (iv) $\mathtt{It}(g\circ x,1)$, respectively. □

Given a morphism $\beta : C\to L$ then (again by analogy with the natural numbers) define the *β-iterate of f* by $f^\beta = \mathtt{It}(f)\circ\langle\beta,1\rangle$. Of course, if there is a parametrised NNO then $f^\beta = f^{\#\circ\beta}$.

Proposition 5.4 $\mathtt{tail}^{\mathtt{id}} = \mathtt{nil}$. *Hence, if there is a NNO then* $\mathtt{tail}^{\#} = \mathtt{nil}$.
Proof It suffices to prove that $\mathtt{tail}^{\mathtt{id}} = \mathtt{foldr}(\mathtt{nil},\pi')$. The $\mathtt{nil}$ condition is trivial, while that for $\mathtt{cons}$ is the commutativity of

$$\begin{tikzcd}
L \arrow[d, "\delta"] & & & & A\times L \arrow[llll, "\mathtt{cons}"'] \arrow[dl, "\delta"'] \arrow[ddd, "1\times\delta"] \\
L^2 \arrow[dddd, "\mathtt{It}(\mathtt{tail})"] & & (A\times L)\times L \arrow[ll, "\mathtt{cons}\times 1"'] \arrow[d, "1\times\mathtt{tail}"] & (A\times L)^2 \arrow[l, "1\times\mathtt{cons}"'] \arrow[dl, "1\times\pi'"] & \\
 & & (A\times L)\times L \arrow[d, "\pi'\times 1"] & & \\
 & & L^2 \arrow[dll, "\mathtt{It}(\mathtt{tail})"] & & A\times L^2 \arrow[ll, "\pi'"'] \arrow[d, "1\times\mathtt{It}(\mathtt{tail})"'] \\
L & & & & A\times L \arrow[llll, "\pi'"]
\end{tikzcd}$$

The non-trivial pentagon commutes by Lemma 0.5.3(ii). □

Theorem 5.5 *If* $f^\beta = m\circ i$ *for some subobject* $m : F\to C$ *fixed by* f *then* $m = \mathtt{fix}(f)$ *and* $i\circ m = 1$. *If* f^β *is also an invariant for* f *then* f *converges with* $\mathtt{inv}(f) = i$.
Proof If x is fixed by f then

$$x = f^\beta\circ x = m\circ i\circ x$$

by Lemma 0.5.3(iii). Hence $m = \mathtt{fix}(f)$ is the fix object. Now $f^\beta\circ m = m$ by Lemma 0.5.3(iii) whence $i\circ m = 1$ since m is a monomorphism. Finally, if g is an invariant for f then $g\circ f^\beta = g\circ\pi'\circ\langle\beta,1\rangle = g$ by Lemma 0.5.3(iv). Thus g factors through i, uniquely since i is a split epimorphism. Hence $i = \mathtt{inv}(f)$ if it is an invariant for f which follows from that of f^β. □

Certainly f^β is an invariant for f if the successor operation can be represented on L (see Section 0.7) but the general question is still open. For our particular applications we can simulate the successor, as required, by picking up a copy of A elsewhere. Define $\mathtt{share} : L^2 \to L^2$ by

$$L^2 \xrightarrow{d \circ (1\times\mathtt{pop})} L + (L\times A\times L) \xrightarrow{[\langle 1, \mathtt{nil}\rangle, \langle \mathtt{cons} \circ c \circ \pi, \mathtt{cons} \circ \pi' \rangle]} L^2$$

It appends a copy of the head of the second list, if it has one, onto the first, while leaving the second list intact. Given C define $\mathtt{share}' = (1\times c) \circ (\mathtt{share}\times 1) \circ (1\times c) : L\times C\times L \to L\times C\times L$.

Lemma 5.6 $\mathtt{share} \circ \langle \mathtt{tail}, 1\rangle = \delta : L \to L^2$

$$\mathtt{It}(\mathtt{shunt}(\alpha)) \circ \mathtt{share}' = \mathtt{It}(\mathtt{shunt}(\alpha)) \circ (1\times \mathtt{shunt}(\alpha)) : L\times C\times L \to L\times C\times L$$

Proof Compose both sides with the inclusions $\mathtt{nil}$ and $\mathtt{cons}$ for the first equation, and with $1\times 1\times \mathtt{nil}$ and $1\times 1\times \mathtt{cons}$ for the second. □

Lemma 5.7 *If α' and $\beta' : A\times D \to D$ are left A-actions and $h : C \to D$ is an A-action homomorphism (i.e. $h \circ \alpha' = \beta' \circ (1\times h)$) then*

$$\begin{array}{ccc} L\times C & \xrightarrow{\mathtt{foldr}(\alpha')} & C \\ {\scriptstyle 1\times h}\downarrow & & \downarrow{\scriptstyle h} \\ L\times D & \xrightarrow[\mathtt{foldr}(\beta')]{} & D \end{array}$$

In particular, $\pi' \circ \mathtt{It}(\mathtt{shunt}(\alpha)) = \mathtt{It}(\mathtt{tail}) \circ (1\times \pi') : L\times C\times L \to L$

Proof Both sides of the square equal $\mathtt{foldr}(h, \beta')$. Now take $h = \pi' : C\times L \to L$. □

6 Comparison of Stacks

Theorem 6.1 *Let C be a polynomial category. Then a parametrised stack for an object A is a convergent stack. The converse holds if C has all equalisers.*

Proof Let **L** be a parametrised stack for A. We will use Lemma 0.5.5 to show that $\mathtt{shunt}(\alpha)$ converges with $\beta = \pi' : C\times L \to L$.

$$\begin{aligned} \pi' \circ \mathtt{shunt}(\alpha)^{\pi'} &= \pi' \circ \mathtt{It}(\mathtt{shunt}(\alpha))\langle \pi', 1\rangle \\ &= \mathtt{It}(\mathtt{tail}) \circ (1\times \pi') \circ \langle \pi', 1\rangle \\ &= \mathtt{It}(\mathtt{tail}) \circ \delta \circ \pi' \\ &= \mathtt{tail}^{\mathtt{id}} = \mathtt{nil} \end{aligned}$$

The second equation holds by Lemma 0.5.6 while the fourth is just Lemma 0.5.4. Thus $\mathtt{shunt}^{\pi'}$ factors through $\langle 1, \mathtt{nil}\rangle$ which equals $\mathtt{fix}(\mathtt{shunt}(\alpha))$ since it is fixed by

$\mathtt{shunt}(\alpha)$. It remains to prove that $\mathtt{shunt}(\alpha)^{\pi'}$ is an invariant for $\mathtt{shunt}(\alpha)$.

$$
\begin{aligned}
\mathtt{shunt}(\alpha)^{\pi'} \circ \mathtt{shunt}(\alpha) &= \mathtt{It}(\mathtt{shunt}(\alpha)) \circ \langle \pi', 1 \rangle \circ \mathtt{shunt}(\alpha) \\
&= \mathtt{It}(\mathtt{shunt}(\alpha)) \circ (1 \times \mathtt{shunt}(\alpha)) \circ \langle \pi' \circ \mathtt{shunt}(\alpha), 1 \rangle \\
&= \mathtt{It}(\mathtt{shunt}(\alpha)) \circ \mathtt{share}' \circ \langle \mathtt{tail} \circ \pi' \circ, 1 \rangle \\
&= \mathtt{It}(\mathtt{shunt}(\alpha)) \circ \langle \pi', 1 \rangle = h
\end{aligned}
$$

where the third step follows from Lemma 0.5.6 and from π' being a loop morphism from $\mathtt{shunt}(\alpha)$ to $\mathtt{tail}$.

For the converse, assume that $\mathbf{L}$ is a convergent stack. Define

$$
\mathtt{foldr}(\alpha') = \mathtt{foldl}(\alpha) \circ (1 \times \mathtt{rev}) \circ c : L \times C \rightarrow C
$$

Then

$$
\mathtt{foldr}(x, \alpha') = \mathtt{foldr}(\alpha') \circ (1 \times x) : L \times B \rightarrow C
$$

The compatability of $\mathtt{foldr}(x, \alpha')$ with $\mathtt{nil}$ is straightforward. For cons consider the following diagram

$$
\begin{tikzcd}
A \times L \times C \arrow[rrr, "\mathtt{cons} \times 1"] \arrow[d, "1 \times c"] & & & L \times C \arrow[d, "c"'] \\
A \times C \times L \arrow[r, "c \times 1"] \arrow[d, "1 \times 1 \times \mathtt{rev}"] & C \times A \times L \arrow[rr, "1 \times \mathtt{cons}"] \arrow[d, "1 \times 1 \times \mathtt{rev}"] & & C \times L \arrow[d, "1 \times \mathtt{rev}"'] \\
A \times C \times L \arrow[r, "c \times 1"] \arrow[d, "1 \times \mathtt{foldl}(\alpha)"] & C \times A \times L \arrow[r, "1 \times c"] & C \times L \times A \arrow[r, "1 \times \mathtt{snoc}"] \arrow[d, "\mathtt{foldl}(\alpha) \times 1"'] & C \times L \arrow[d, "\mathtt{foldl}(\alpha)"'] \\
A \times C \arrow[rr, "c"'] & & C \times A \arrow[r, "\alpha'"'] & C
\end{tikzcd}
$$

To show that the last square commutes it suffices to prove that $\mathtt{foldl}(\alpha) \circ (1 \times \mathtt{snoc})$ is an invariant for $\mathtt{shunt}(\alpha) \times 1$ since then it factors through $\mathtt{foldl}(\alpha)$ by some morphism, which is revealed to be α on composing with $1 \times \mathtt{nil} \times 1$. The invariance follows from the commutativity of

$$\begin{tikzcd}
C\times A\times L\times A \arrow[rr, "1\times\mathtt{cons}\times 1"] \arrow[d, "\alpha\times 1\times 1"'] \arrow[dr, "1\times 1\times\mathtt{snoc}"] & & C\times L\times A \arrow[dr, "1\times\mathtt{snoc}"] & \\
C\times L\times A \arrow[dr, "1\times\mathtt{snoc}"'] & C\times A\times L \arrow[rr, "1\times\mathtt{cons}"] \arrow[d, "\alpha\times 1"] & & C\times L \arrow[d, "\mathtt{foldl}(\alpha)"] \\
& C\times L \arrow[rr, "\mathtt{foldl}(\alpha)"'] & & C
\end{tikzcd}$$

whose upper square commutes by Lemma 0.4.3 (vii).

We will use equalisers to show that $\mathtt{foldr}(x,\alpha')$ is the unique map making the desired diagram commute. It is still an open question whether they are necessary for the result. If there are two such maps then their equaliser is a subobject $m : R \rightarrow L\times B$ compatible with the parametrised action, i.e. we have morphisms r_0 and r_1 making the following diagram commute

$$\begin{tikzcd}
& R \arrow[d, "m"] & A\times R \arrow[l, "r_1"'] \arrow[d, "1\times m"] \\
B \arrow[ur, "r_0"] \arrow[r, "\langle\mathtt{nil},1\rangle"'] & L\times B & A\times L\times B \arrow[l, "\mathtt{cons}\times 1"]
\end{tikzcd}$$

Now the following diagram commutes

$$\begin{tikzcd}
B\times L \arrow[r, "r_0\times\mathtt{rev}"] \arrow[d, "\langle\mathtt{nil},1\rangle\times 1"'] & R\times L \arrow[r, "\mathtt{foldl}(r_1\circ c)"] \arrow[d, "m\times 1"'] & R \arrow[d, "m"] \\
L\times B\times L \arrow[r, "1\times 1\times\mathtt{rev}"'] & L\times B\times L \arrow[r, "\mathtt{foldl}((\mathtt{cons}\times 1)\circ c)"'] & B\times L
\end{tikzcd}$$

since m is an A-action homomorphism. Further, the lower path equals c since $\mathtt{rev}\circ\mathtt{rev} = 1$. Thus the monomorphism m is also an split epimorphism, and hence an isomorphism, completing the proof. □

7 Terminating Loops

When $(N, 0, S)$ is a parametrised NNO then we can describe those loops which converge by becoming fixed after a specified number of iterations.

Definition 7.1 *A loop $f : C \to C$ is* bound *if there is a* bounding morphism $\beta : C \to N$ *such that $f^\beta : C \to C$ is fixed by f. If f^β also factors through a morphism $m : F \to C$ fixed by f then it* terminates.

Of course, Lemma 0.5.5 guarantees that $m = \mathtt{fix}(f)$. That the choice of bounding map doesn't affect the value of the resulting fixpoint is shown by the following

Lemma 7.2 *If $\beta, \gamma : C \to N$ are both bounding maps for the loop f then $f^\beta = f^\gamma$.*
Proof

$$\begin{aligned} f^\beta &= \mathtt{It}(f) \circ \langle \gamma, f^\beta \rangle \\ &= \mathtt{It}(f) \circ (1 \times \mathtt{It}(f)) \circ \langle \gamma, \beta, 1 \rangle \\ &= \mathtt{It}(f) \circ \langle \gamma + \beta, 1 \rangle \\ &= \mathtt{It}(f) \circ \langle \beta + \gamma, 1 \rangle \\ &= f^\gamma \end{aligned}$$

The first and last equalities follow from Lemma 0.5.3(iii) while the third holds since both sides equal $\mathtt{It}(\mathtt{It}(f), f)$. □

Theorem 7.3 *Terminating loops converge.*
Proof Let $f : C \to C$ terminate with bounding morphism β. By Lemma 0.5.5 it suffices to prove that f^β is an invariant for f.

$$\begin{aligned} f^\beta \circ f &= \mathtt{It}(f)(1 \times f)\langle \beta \circ f, 1 \rangle \\ &= \mathtt{It}(f)\langle S \circ \beta \circ f, 1 \rangle \\ &= \mathtt{It}(f)\langle \beta, 1 \rangle \end{aligned}$$

The second equality holds by Lemma 0.5.3(ii). For the third equality note that $S \circ \beta \circ f$ is a bounding map for f since the lefthand side is fixed by f. Now apply Lemma 0.7.2.
□

If f^β (for $\beta : C \to N$) is an invariant for f then f is a *contraction* [2]. Thus the terminating loops are *contractions* (with the converse following directly).

A convergent stack may satisfy the stronger condition that $\mathtt{shunt}(\alpha)$ always terminates, or equivalently, that `tail` terminates. Define a *terminating stack* on A to be a stack whose `tail` terminates with fixpoints given by `nil`. These are exactly the *pre-recursives* on $(A, 1)$ in [2].

Now the previous comparison theorem may be expanded to:

Theorem 7.4 *If C is polynomial with a parametrised NNO then we have the following implications for stacks on A:*

$$\begin{aligned} \textit{parametrised stack} &\Rightarrow \textit{initial stack} \\ &\Rightarrow \textit{terminating stack} \\ &\Rightarrow \textit{convergent stack} \end{aligned}$$

If C has all equalisers then they are all equivalent.

This result generalises the work of Cockett [2] who showed the equivalence of the first three of these list concepts when the category $\mathcal{C}$ is a distributive category. The presence of convergent stacks simplifies the proof of this partial result, as well as accounting for tail recursion. One consequence of the result is that in the presence of equalisers, parametrisation of all list objects follows from that of the NNO. At the risk of oversimplifying, it appears that `foldr` produces the cleanest mathematical constructions, `foldl` yields efficient implementations, while terminating stacks are useful for proving properties of lists.

Proof If L is a parametrised stack then it is initial by definition.

If **L** is an initial stack then the bounding map for `tail` is the length # of the list. $\mathtt{tail}^{\#} = \mathtt{nil}$ (by Lemma 0.5.4) which is fixed by `tail`.

If `tail` is a terminating loop with bounding map β then we will show that $\mathsf{shunt}(\alpha) : C \times L \to C \times L$ has bounding map $\beta \circ \pi'$. Now

$$\begin{tikzcd}
C\times L \arrow[r, "{\langle \beta\circ\pi', 1\rangle}"] \arrow[d, "\pi'"'] & N\times C\times L \arrow[r, "\mathtt{It}(\mathsf{shunt}(\alpha))"] \arrow[d, "1\times\pi'"'] & C\times L \arrow[d, "\pi'"] \\
L \arrow[r, "{\langle \beta, 1\rangle}"'] & N\times L \arrow[r, "\mathtt{It}(\mathtt{tail})"'] & L
\end{tikzcd}$$

commutes by Lemma 0.5.6. The bottom/left of the diagram is `nil` by assumption and so the top edge of this square factors through $\langle 1, \mathtt{nil}\rangle$ which is fixed by $\mathsf{shunt}(\alpha)$. Thus $\mathsf{shunt}(\alpha)$ is a terminating loop, and so converges, with fixpoints given by $\langle 1, \mathtt{nil}\rangle$. □

8 Acknowledgements

My thanks go to Robin Cockett for many fruitful discussions. I have also profited from stimulating conversations with John Hughes, Geraint Jones, Ed Kazmierczak, David Murphy, Mary Sheeran, and the members of the Applications of Categories Club.

References

[1] R. Bird and P. Wadler, *Introduction to Functional Programming* International Series in Computer Science, ed: C.A.R. Hoare (Prentice Hall, 1988).

[2] J.R.B. Cockett, List-arithmetic distributive categories: locoi, J. Pure and Appl. Alg. 66 (1990) 1–29.

[3] J.R.B. Cockett, Distributive Theories, in: G. Birtwistle (ed), IV Higher Order Workshop, Banff, 1990, (Springer, 1991).

[4] A note on natural numbers objects in monoidal categories, Studia Logica 48(3) (1989) 389–393.

[5] C.B. Jay, Fixpoint and loop constructions as colimits, preprint.

[6] C.B. Jay, Partial functions, ordered categories, limits and cartesian closure, in: G. Birtwistle (ed), IV Higher Order Workshop, Banff, 1990, (Springer, 1991).

[7] C.B. Jay, Matrices, monads and the Fast Fourier Transform, in preparation.

[8] C.B. Jay, An order-theoretic approach to partial maps, in preparation.

[9] G. Jones, Calculating the Fast Fourier Transform, in: G. Birtwistle (ed), IV Higher Order Workshop, Banff, 1990, (Springer, 1991).

[10] J. Lambek, A fixpoint theorem for complete categories, Math. Zeitschr. 103 (1968), 151–161.

[11] J. Lambek, Fixpoints revisited, preprint.

[12] J. Lambek and P. Scott, Introduction to higher order categorical logic, Cambridge studies in advanced mathematics 7, Cam. Univ. Press (1986).

[13] S. Mac Lane, *Categories for the Working Mathematician* (Springer-Verlag, 1971).

[14] E. Moggi, Computational lambda-calculus and monads, Proceedings Fourth Annual Symposium on Logic in Computer Science (1989) 14–23.

[15] R. Paré and L. Roman, Monoidal categories with natural numbers object, Studia Logica 48(3) (1989).

[16] L. Roman, Cartesian categories with natural numbers object, J. of Pure and Appl. Alg. 58 (1989) 267–278.

[17] M.B. Smyth & G.D. Plotkin, *The Category-theoretic solution of recursive domain equations*, SIAM J. of Comp. 11 (1982).

[18] R.F.C. Walters, Datatypes in distributive categories, Bull. Australian Math. Soc. 40 (1989) 79–82.