# PARAMETRIZED DATA TYPES DO NOT NEED HIGHLY CONSTRAINED PARAMETERS

Michael A. Arbib
Computer and Information Science
and
Ernest G. Manes
Mathematics and Statistics

COINS Technical Report 81-23

(September 1981)

(Revised October 1982)

To appear in the journal Information and Control.

# PARAMETRIZED DATA TYPES DO NOT NEED HIGHLY CONSTRAINED PARAMETERS[1]

Michael A. Arbib
Computer and Information Science

Ernest G. Manes
Mathematics and Statistics
University of Massachusetts
Amherst, MA 01003

## Abstract

We argue that data types may be considered as objects in any suitable category, and need not necessarily be ordered structures or many-sorted algebras. We show that arrays may be specified having as parameter any object from a category $\mathcal{K}$ with finite products and coproducts, if products distribute over coproducts. We extend the Lehmann-Smyth least fixpoint approach to recursively-defined data types by introducing the dual notion of greatest fixpoint, which allows us to define infinite lists and trees without recourse to domains bearing a partial order structure. Finally, we show how the least fixpoint approach allows us to define queues directly in terms of stacks, rather than through a separate equational specification.

---

[1] The research reported in this paper was supported in part by Grant No. MCS 80-05112 from the National Science Foundation.

## 1. What is a Data Type?

The notion "recursively-defined data type" has two meanings: Perhaps the one more common in the literature on abstract specification is that in which it is the individual data type which receives the recursive definition, as in the definition

$$L = 1 + A \times L$$

for (possibly empty) lists of elements from A. It is in this sense that we shall use a recursively-defined data type in this paper. On the other hand, in a language with structured data types as in PASCAL, we may say that it is the family of data types which is recursively defined: we start with certain basic data types, and then recursively define new data types through the use of arrays, records, files, pointers, etc. When we have a scheme for such building, e.g., arrays built from an arbitrary data type, we may speak of the scheme as a parametrized data type. In this paper we build on the work of earlier authors to offer a somewhat eclectic approach to both types of specification which (although using some basic machinery from categorical algebra) seems to be more "intuitive" than other formal approaches, without losing rigor. In this introduction, we give a general overview of the constraints from which we free our data types, and then provide a more detailed development in subsequent sections.

Scott (e.g., [1977]) has argued that recursively-defined data types are to be determined by successive approximation, and should thus be defined in some category, Dom, of ordered structures, such as complete lattices or

ω-cpos. By extension, then, there is a school of thought which views a data type as being simply an object of a suitable category, Dom.

Other authors (e.g. Guttag [1977], Goguen, Wagner, Thatcher and Wright [1975], Liskov and Zilles [1974]) have stressed that a data type encompasses not only a carrier (e.g., the set of integers), but also operations (e.g., addition, test for positivity), and that these operations may involve sets other than the carrier (e.g., Bool = {true, false}). An abstract specification of a data type is then to be given by a set of equations the operations must satisfy (or, more generally, a set of conditional specifications that these and other 'hidden' operations must satisfy). A data type is then to be seen as some generalized variety of many-sorted algebras.

Our general viewpoint will be as follows:

(i) Data types are objects of a suitable category.

(ii) In general, order structure need not be placed on the objects of the category. To the extent that successive approximations determine a recursively-defined data type, they will arise naturally as chains of morphisms in a limit construction (but not in a colimit construction).

(iii) While equational specification is appropriate for certain data types, there will often be direct constructions which are more 'natural' from a programming viewpoint. For example, we shall show how to construct a queue from a stack, rather than giving it a direct equational specification (which would say nothing of the normal FIFO vs. LIFO relation between queue and stack).

We shall give examples of what we mean by (i) in Section 2; shall turn to (ii) in Section 3; and treat (iii) in Section 4. Section 2 is elementary, to give the flavor of our approach; Sections 3 and 4 are more technical in their use of category theory.

## 2. 'Array' as a Parametrized Data Type

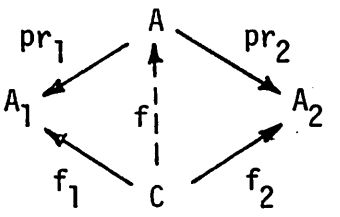
The dictum "data types are objects in a suitable category" has the corollary that "parametrized data types are ways of constructing objects from other objects (the parameters)", it being left open as to what categories the objects belong to. Perhaps as fundamental a parametrized data type as any is the array:

array 1...n of C

is a data type whose carrier is the $n^{th}$ cartesian power of the carrier of the data type C, and which comes equipped with operations for reading and assigning any one of the n components. The point to be stressed is that C need not be a set; it could be a file or record or stack of records of files, etc. In each case, the components of the array are not to be seen as set elements, but rather as structured entities subject to the operations of the data type C.

Category theory is well-suited to handle this sort of situation. For example, the familiar definition of Cartesian product, $A_1 \times A_2 = \{(a_1, a_2) \mid a_1 \in A_1, a_2 \in A_2\}$, of two sets A and B, receives the following generalization:

Definition: In a category $\mathcal{K}$, an object A equipped with two morphisms $pr_1 : A \rightarrow A_1$ and $pr_2 : A \rightarrow A_2$ is said to be the product of $A_1$ and $A_2$ if for ever pair of morphisms $f_1 : C \rightarrow A_1$ and $f_2 : C \rightarrow A_2$ there is a unique $f : C \rightarrow A$ such that $pr_i \cdot f = f_i$ as shown in the 'commutative diagram'.




[In the category Set of sets and functions, take $A = A_1 \times A_2$, $pr_j(a_1,a_2) = a_j$, $f(c) = (f_1(c), f_2(c))$.]

Thinking of the notion of product as "array in miniature", we make the following observations which bear upon the general notion of a parametrized data type:

(a) The definition makes sense in every category.

(b) However, in a given category it is a matter to be determined whether (i) every pair of objects has a product; or (ii) if a given pair of objects has a product.

(c) If the product of two objects exists, it is unique up to unique isomorphism.

Combining observations (a) and (b), it will often prove convenient, when giving a general categorical construction (parametric data type specification), to place restrictions on the category $\mathcal{K}$ which guarantee that the construction goes through for all choices of the objects (parameters). To illustrate, we now give a general definition of array 1...n of C, first working with Set, and then generalizing the construction so that it works in a broad class of categories $\mathcal{K}$.

Consider, then, the data type consisting of arrays of length n with entries from the set C. It is given by the set $C^n$ together with two maps

as follows, where [n] denotes the set {1,...,n}:

$$\begin{aligned} C^n \times [n] &\longrightarrow C, \qquad (x,i) \mapsto x[i] = pr_i(x) \\ C^n \times C \times [n] &\longrightarrow C^n, \quad ((c_1,\ldots,c_n),c,i) \mapsto (c_1\ldots,c_{i-1},c,c_{i+1},\ldots,c_n) \end{aligned} \tag{1}$$

where the latter makes possible the assignment x[i] := c.

We may say that if C is an object of the category Set, then array 1..n of C is an object of the functor category (see, e.g., Arbib and Manes [1975], p. 153) $\underline{\underline{Set}}^{\Delta}$ where Δ is the category

$$\begin{array}{ccc} \bullet & \longrightarrow & \bullet \\ & & \\ \bullet & \longrightarrow & \bullet \end{array} \tag{2}$$

with 4 objects and 2 non-identity morphisms as shown. We wish to view array 1..n of (•) as a functor X from Set to $\underline{\underline{Set}}^{\Delta}$. Its action on objects is given by (1), while its action on morphisms sends f : C → D to Xf : XC → XD with the diagrams

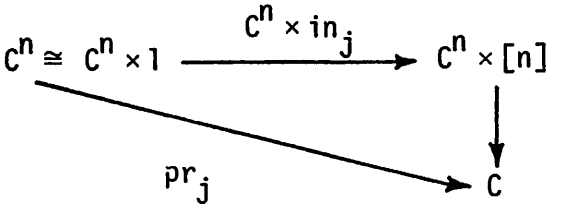
$$\begin{array}{ccc} C^n \times [n] & \longrightarrow & C \\ {\scriptstyle f^n \times id_n}\downarrow & & \downarrow{\scriptstyle f} \\ D^n \times [n] & \longrightarrow & D \end{array} \qquad \begin{array}{ccc} C^n \times C \times [n] & \longrightarrow & C^n \\ {\scriptstyle f^n \times f \times id_n}\downarrow & & \downarrow{\scriptstyle f^n} \\ D^n \times D \times [n] & \longrightarrow & D^n \end{array}$$

which do commute for any function f.

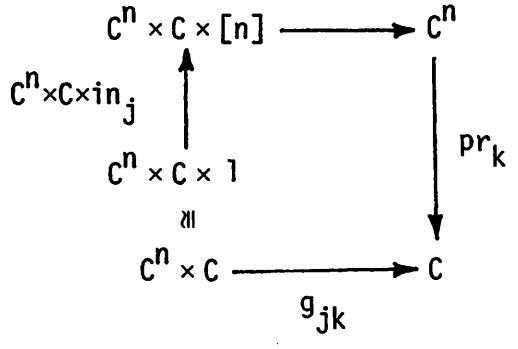
Clearly, the above definition only uses some very general properties of Set. Let us then define X = array 1..n of (•) in the following general setting:

$$\begin{aligned} &\mathcal{K} \text{ has finite products and coproducts} \\ &\text{(including a terminal object, 1), and} \\ &A \times - \text{ preserves coproducts for each object A of } \mathcal{K}. \end{aligned} \tag{3}$$

Generalizing our previous notation, write [n] for the coproduct of n copies of 1. We first define $C^n$ as the product of n copies of C, and then define $C^n \times [n] \longrightarrow C$ by




while $C^n \times C \times [n] \longrightarrow C^n$ is defined by




where $g_{jk} = \begin{cases} pr_2 & \text{if } j = k \\ pr_k \cdot pr_1 & \text{if not.} \end{cases}$

Extending the definition to morphisms, array 1..n of (f), is a straightforward exercise, and is omitted.

We have thus defined the parametrized data type array 1..n of ( ) in terms of a functor from $\mathcal{K}$ to $\mathcal{K}^{\Delta}$ which is defined for any category satisfying the conditions (3), with $\Delta$ the diagram category of (2). The parameter C in array 1..n of C can then be any object of any such category $\mathcal{K}$, and since array 1..n of C lives in $\mathcal{K}^{\Delta}$, all the operations of $\mathcal{K}$ are "packed" within the specification.

It seems to us that, in this case, any further restrictions upon $\mathcal{K}$ would be gratuitous. We emphasize, of course, that other specifications may

require greater or lesser stringency in choosing the categories whose objects serve as parameters. We may contrast this with the definition of Thatcher, Wagner and Wright [in press]:

> A parametrized specification consists of a parameter signature $\Sigma$, parameter conditions E, resultant signature $\Sigma'$ (with $\Sigma \subseteq \Sigma'$) and resultant axioms E'. ... The specified parametrized type is the functor
>
> $$F_E : \underline{\underline{Alg}}_{\Sigma,E} \longrightarrow \underline{\underline{Alg}}_{\Sigma',E'}$$
>
> which is obtained from the functor $F : \underline{\underline{Alg}}_{\Sigma} \longrightarrow \underline{\underline{Alg}}_{\Sigma',E'}$ which takes each $\Sigma$-algebra A to the $(\Sigma',E')$-algebra freely generated by A, restricting it to algebras satisfying E.

In contrast to our definition of arrays, we may note:

(a) That the specified parametrized type is a single functor $\mathcal{K} \longrightarrow \mathcal{K}'$ for a fixed $\mathcal{K}$, rather than a general recipe for constructing functors applicable to a wide class of categories $\mathcal{K}$; and

(b) The category $\mathcal{K}$ is restricted to be of the specific form $\underline{\underline{Alg}}_{\Sigma,E}$, the category of $(\Sigma,E)$-algebras.

## 3. Fixpoint Approaches to Data Types[1]

Where Scott [1977] has introduced recursively-defined data types as least fixed points of continuous functionals and Goguen, Thatcher, Wagner and Wright [1975] have defined such data types as initial algebras, Lehmann and Smyth [1981] and others have constructed such data types as 'least fixpoints' of functorial equations $XQ \cong Q$ for $X : \mathcal{K} \longrightarrow \mathcal{K}$ an endofunctor. The present section continues the work of Lehmann and Smyth by exploring the dual construction of a 'greatest fixed point'. In particular, we show that

[1] A preliminary version of this section was presented as Arbib and Manes [1980b].

many interesting data types are defined as greatest fixpoints of functorial equations $XQ \cong Q$ that live in Set. Successive approximations are given by chains of maps in Set; thus no additional order structure need be placed on the objects of the category.

An isomorphism $\mu : XL \to L$ is said to be a least fixpoint of X if it has the universal property

$$\begin{array}{ccc} XL & \xrightarrow{\mu} & L \\ {\scriptstyle X\phi}\downarrow & & \downarrow{\scriptstyle \phi} \\ XQ & \xrightarrow[\delta]{} & Q \end{array} \qquad (1)$$

that for any $\delta : XQ \to Q$ (not necessarily an isomorphism) there is a unique $\phi$ such that (1) commutes.

We here introduce a dual concept: We say that an isomorphism $M : G \to XG$ is a greatest fixpoint of X if it satisfies the dual universal property

$$\begin{array}{ccc} Q & \xrightarrow{\Delta} & XQ \\ {\scriptstyle \psi}\downarrow & & \downarrow{\scriptstyle X\psi} \\ G & \xrightarrow[M]{} & XG \end{array} \qquad (2)$$

that for arbitrary $\Delta : Q \to XQ$, there is a unique $\psi : Q \to G$ such that (2) commutes.

## 3.1. Examples of Greatest Fixpoints

Given an input alphabet A and an output set Y, consider the following specification of the 'state' of an automaton: a state is an output together with a next-state function from the inputs. In functorial form,

$$XQ \cong Q, \quad XQ = Y \times [A \to Q]. \qquad (3)$$

In the category Set, the least fixpoint is the empty set. On the other hand, (3) has $[A^* \to Y]$ as greatest fixpoint, and the universal property (2) reads as follows:

$$\begin{array}{ccc} Q & \xrightarrow{\Delta = \binom{\beta}{\gamma}} & Y \times [A \to Q] \\ {\scriptstyle\sigma}\downarrow & & \downarrow{\scriptstyle X\sigma} \\ [A^* \to Y] & \xrightarrow[M]{} & Y \times [A \to [A^* \to Y]] \end{array} \qquad (4)$$

Here, the isomorphism M sends a map $f : A^* \to Y$ to the pair $(f(\Lambda), fL_{(\cdot)})$ where for each a in A, $fL_a$ is the map $w \mapsto f(aw)$. Any $\Delta : Q \to Y \times [A \to Q]$ amounts to a state-transition $\gamma : Q \times A \to A$ together with an output map $\beta : Q \to Y$, and (4) then unpacks as

$$\sigma(q)(\Lambda) = \beta(q)$$
$$\sigma(q)(aw) = \sigma(\gamma(q,a))(w)$$

Which defines $\sigma(q)$ as the observability map of q, as in automata theory.

For the second example, assume given a function $f : A \to A + B$ and consider the data type $A^*B + A^\infty$ that arises in defining the historical iterate $g : A \to A^*B + A^\infty$ of f. Here $g(a) = (a_1,\ldots,a_n,b)$ in $A^*B$ if iteration terminates after passing through the sequence $(a_1,\ldots,a_n)$ of 'states' in A, before exit with value b; while g(a) is the infinite sequence of 'states' in A obtained by repeated application of f if exit never occurs. This type satisfies the fixpoint equation

$$XQ \cong Q, \quad XQ = (A \times Q) + B \qquad (5)$$

whose least fixpoint is only $A^*B$ but whose greatest fixpoint is indeed $A^*B + A^\infty$, with the isomorphism

$$M : A^*B + A^\infty \longrightarrow (A \times (A^*B + A^\infty)) + B$$

sending b in A*B to b in B; awb in A*B to (a,wb) in $A \times A^*B$; and aw in $A^\infty$ to (a,w) in $A \times A^\infty$. Now consider the least fixpoint property:

$$\begin{array}{ccc} Q & \xrightarrow{\Delta} & (A \times Q) + B \\ g \downarrow & & \downarrow g \\ A^*B + A^\infty & \longrightarrow & A \times (A^*B + A^\infty) + B \end{array} \qquad (6)$$

We may (Arbib and Manes, 1980a) decompose $\Delta$ into two <u>partial functions</u> $\Delta_1$ with domain $\Delta^{-1}(A \times Q)$ and $\Delta_2$ with domain $\Delta^{-1}(B)$. Then we define the partial function $\Delta_1^{(n)}: Q \rightharpoonup A^* \times Q$ inductively as follows:

$$\Delta_1^{(0)}(q) = (\Lambda, q)$$

and for $n \geq 1$,

$$\Delta_1^{(n)}(q) = \begin{cases} (a_1 \ldots a_n, q_n) & \text{if } \Delta_1^{(n-1)} = (a_1 \ldots a_{n-1}, q_{n-1}) \text{ is defined,} \\ & \text{and } \Delta_1(q_{n-1}) = (a_n, q_n) \text{ is defined} \\ \text{undefined} & \text{if not.} \end{cases}$$

It is then clear that

$$g(q) = \begin{cases} a_1 a_2 \ldots a_n \ldots & \text{in } A^\infty \text{ if } \Delta_1^{(n)}(q) = (a_1 \ldots a_n, q_n) \text{ is defined} \\ & \text{for every } n \geq 1. \\ a_1 a_2 \ldots a_{n-1} b & \text{in } A^*B \text{ if } \Delta_1^{(n-1)}(q) = (a_1 \ldots a_{n-1}, q_{n-1}) \text{ is} \\ & \text{defined, and } \Delta_2(q_{n-1}) = b. \end{cases}$$

In particular, if we replace $f : A \rightharpoonup A + B$ by $\Delta = (\mathrm{diag} + B) \circ f : A \rightharpoonup A + B \longrightarrow A \times A + B$, (where $\mathrm{diag}(a) = (a,a)$) then we do recapture the desired historical iterate g, and (6) gives the recursive definition

$$g(a) = \begin{cases} b & \text{if } f(a) = b \text{ in } B \\ f(a) \cdot g(f(a)) & \text{otherwise.} \end{cases}$$

These examples suggest that much more can be done in the category of sets than was previously believed. With $B = \emptyset$ in (6), we see that $A^\infty$ is the greatest fixpoint of $QX = A \times Q$ in Set, countering Scott's claim that $A^\infty$ can best be constructed as a topological or order-theoretic 'completion' of A*. To see why a category Dom of ordered domains is not required note that the important idea of 'finite approximation' in Scott's work arises naturally in the category of sets in view of the way inverse limits are constructed in that category:

Under suitable conditions on X (reviewed in the next subsection) the least fixpoint of X is given by the colimit construction

$$L = \operatorname*{colim}_n \left(X^n 0 \xrightarrow{X^n t} X^{n+1} 0\right) \tag{7}$$

where 0 is initial in the category, and $t : 0 \to X0$ is the unique map; and we now observe that many functors satisfy the dual condition which yields the greatest fixpoint as the limit (i.e., inverse limit)

$$G = \lim_n \left(X^{n+1} 1 \xrightarrow{X^n u} X^n 1\right) \tag{8}$$

where 1 is terminal in the category and $u : X1 \to 1$ is the unique map.

In the category of sets, an element q of G is represented by a sequence $q_n \in X^n 1$ in which '$q_n$ approximates $q_{n+1}$'. For example, if (8) arises from (5), write $1 = \{\perp\}$. If $q = a_1 \ldots a_k b \in A^*B$ then $q_0 = \perp$, $q_1 = a_1$, $q_2 = a_1 a_2$, ..., $q_k = a_1 \ldots a_k$, $q_{k+1} = a_1 \ldots a_k b = q_{k+2} = \ldots$, whereas $q_n$ is similarly defined (but not ultimately constant) when $q \in A^\infty$.

After a summary of functorial results (mostly known) in 3.2 we shall present further examples in the category of sets in 3.3, with particular emphasis on finite approximations of infinite trees.

## 3.2 Functorial Fixpoints

An ω-chain in a category $\mathcal{K}$ is a diagram of form

$$\longrightarrow K_{n+1} \longrightarrow K_n \longrightarrow \cdots \longrightarrow K_2 \longrightarrow K_1 \longrightarrow K_0.$$

A functor $\mathcal{K} \longrightarrow \mathcal{L}$ is continuous if it preserves limits of ω-chains. Dually, such a functor is co-continuous if it preserves colimits of ω-cochains (so that our co-continuous functors are what others have called continuous). A functor simultaneously continuous and co-continuous we shall call bicontinuous. The following result is standard category theory:

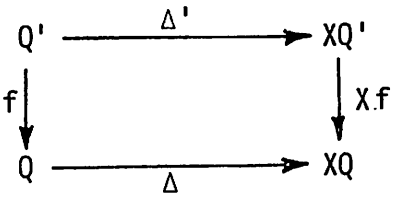
Theorem 1: Every pointwise product of continuous functors is continuous. Every pointwise coproduct of co-continuous functors is co-continuous. The identity functor and all constant functors are bicontinuous. □

For the balance of this section we fix a functor $X : \mathcal{K} \longrightarrow \mathcal{K}$ and assume $\mathcal{K}$ has an initial object 0, a terminal object 1 and whatever limits of ω-chains and colimits of ω-cochains are needed.

An X-dynamics (Arbib and Manes, 1974) is a pair $(Q,\delta)$ with $\delta : XQ \rightarrowtail Q$ and the category of all X-dynamics with morphisms

$$\begin{array}{ccc} XQ & \xrightarrow{\ \delta\ } & Q \\ {\scriptstyle Xf}\downarrow & & \downarrow{\scriptstyle f} \\ XQ' & \xrightarrow[\ \delta'\ ]{} & Q' \end{array} \qquad (9)$$

is written Dyn(X). An X-codynamics is a pair $(Q,\Delta)$ with $\Delta : Q \rightarrowtail XQ$, and the category of all X-codynamics with morphisms as in (10)




(10)

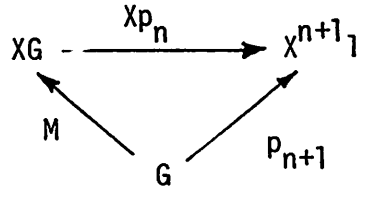
is written Codyn(X). A fixpoint of X is a pair $(Q,\delta)$ with $\delta : XQ \to Q$ an isomorphism. In this case $(Q,\delta)$ is an X-dynamics and $(Q,\delta^{-1})$ is an X-codynamics.

The results for Dyn(X) in Theorems 2 and 3 are from the literature; the results for CoDyn(X) follow simply by duality but appear not to have been noted before for Set. Related concepts and transfinite versions of Theorem 3 below were extensively studied by a number of workers in Prague in a series of papers initiated by Koubek [1971], and surveyed by Adámek and Trnkova [1980]. The early result cited in the next theorem was in a different context.

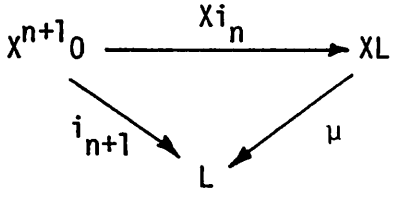
Theorem 2 (attributed to Lambek in Barr [1970]): If Dyn(X) has an initial object, it is an isomorphism. If CoDyn(X) has a terminal object it is an isomorphism. □

This allows us to simplify (1) and (2) by dropping the isomorphism condition. The least fixpoint of X is the initial object of Dyn(X); the greatest fixpoint of X is the terminal object of CoDyn(X).

Theorem 3 (Adámek and Koubek [1979], Lehmann and Smyth [1981]): If X is continuous and G is the limit of (8) with projections $p_n : G \to X^{n}1$ then there exists unique M such that

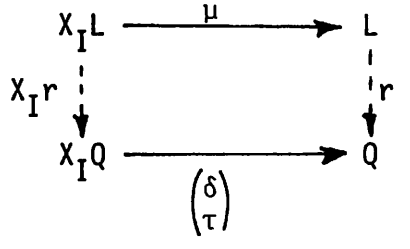


and (G,M) is the terminal object of $\underline{\underline{CoDyn}}(X)$ (and hence is the greatest fixpoint of X). Dually, if X is co-continuous and L is the colimit of (7) with injections $i_n : X^n 0 \to L$, there exists unique $\mu$ such that




and $(L,\mu)$ is the initial object of $\underline{\underline{Dyn}}(X)$ (and so is the least fixpoint of X). □

In this context, it is interesting to recall our original motivation for the study of $\underline{\underline{Dyn}}(X)$ (Arbib and Manes, 1974). We may view an X-dynamics $(Q,\delta)$ equipped with an 'initial state map' $\tau : I \to Q$ as a map $QX + I \xrightarrow{\binom{\delta}{\tau}} Q$. Let us use $X_I$ for the functor $QX_I = QX + I$. We say that $X : \mathcal{K} \to \mathcal{K}$ is a recursion process (or input process, or varietor) if $X_I$ has a least fixpoint for every I in $\mathcal{K}$, and we then refer to the unique r defined by




as the reachability map of the 'initialized machine' $(Q,\delta,I,\tau)$. We thus have

Corollary: Let $\mathcal{K}$ have binary coproducts and $\omega$-colimits. Then every continuous X is a recursion process.

Proof: Just apply the above result to $X_I$, using Theorems 1 and 3. □

## 3.3 Infinite Trees in the Category of Sets

Theorem 4: For functors $\underline{\underline{\text{Set}}} \to \underline{\underline{\text{Set}}}$, any finite product of co-continuous functors is co-continuous and any coproduct of continuous functors is continuous. Hence bicontinuous functors are closed under finite products and arbitrary coproducts. □

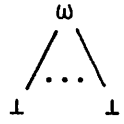
Let $\Omega$ be an operator domain, that is, $\Omega$ is a disjoint sequence $(\Omega_n)$ of sets. $X_\Omega : \underline{\underline{\text{Set}}} \to \underline{\underline{\text{Set}}}$ is defined as

$$X_\Omega = \coprod_{n \geq 0} \Omega_n \times (-)^n \tag{12}$$

Then $X_\Omega$ is bicontinuous.

The least fixpoint of X is the set of all finitely branching trees in which n-ary branch nodes are labelled by an element of $\Omega_n$ (so that all leaves are labelled by elements of $\Omega_0$).

To describe the greatest fixpoint, let T be the set of all finitely branching trees in which n-ary branch nodes for $n \geq 1$ are labelled by elements of $\Omega_n$ but leaves are labelled by elements of $\Omega_0$ or by $\perp$. We shall use $\{\perp\}$ as the terminal object of $\underline{\underline{\text{Set}}}$. For $r, s \in T$ write $r \Rightarrow s$ (r produces s) if s is obtained from r by substituting for each leaf $\perp$ in r (if any) a tree of form




(13)

for $\omega$ in some $\Omega_m$. (If $m = 0$ the substitution tree is just an element of $\Omega_0$). The greatest fixpoint G is the set of all sequences $(r_n)$ in which $r_0 = \perp$ and $r_n \Rightarrow r_{n+1}$. To see the isomorphism $M : G \longrightarrow X_\omega G$, observe

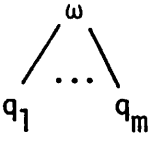
that given $(r_n)$ in G, $r_1$ has form (13) for some $\omega \in \Omega_m$ and that the subsequent evolution of the $i^{th}$ leaf $\perp$ is a sequence $(s_n^i)$ in G;

$r \longleftrightarrow (\omega, s^1, \ldots, s^m)$ describes M.

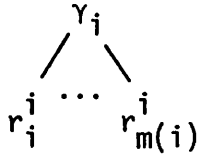
In the universal property (2), $\psi$ is governed by the interesting recursive equation

$$M(\psi(q)) = \begin{array}{c} \omega \\ \psi(q_1) \cdots \psi(q_m) \end{array}$$

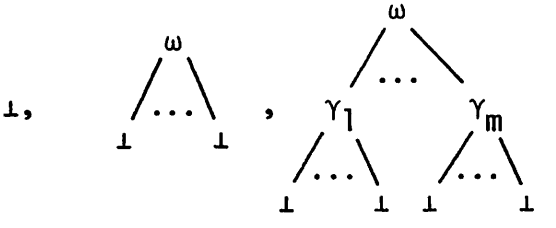
where $\Delta(q) =$



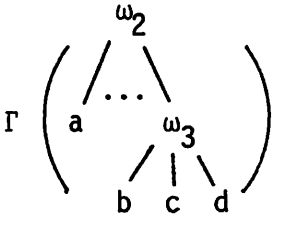

Thus, if $\Delta(q_i) =$



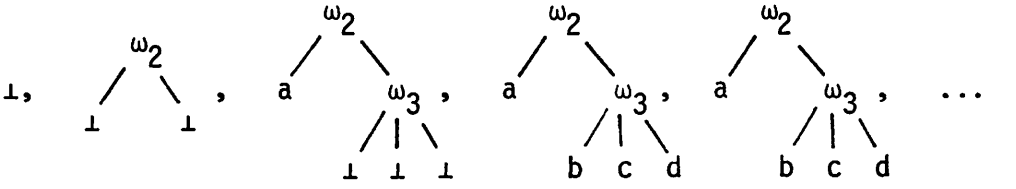

the first three entries of $\psi(q)$ are




If both (7) and (8) are defined for a given X, then by (1) and (2) there is a unique $\Gamma : L \longrightarrow G$ such that $X\Gamma = M\Gamma\mu$. Here, $\Gamma$ embeds each tree in L as an ultimately constant sequence. For example

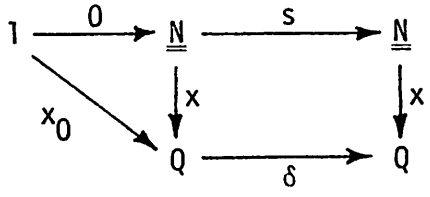



embeds as (15)



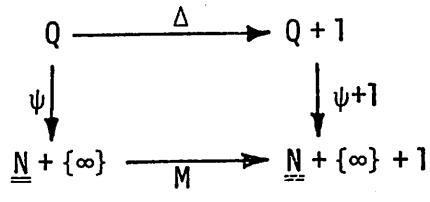
The example culminating in (6) is the special case $\Omega_0 = B$, $\Omega_1 = A$, all other $\Omega_m$ empty.

Further specialization occurs when A, B each have one element. Here the least fixpoint is the Peano natural numbers, the universal property (1) being the principle of simple recursion:




$$x(0) = x_0$$
$$x(n+1) = \delta(x(n))$$

A codynamics $Q \longrightarrow Q + 1$ amounts to a partial function from Q to Q. The greatest fixpoint is $\underline{\underline{N}} + \{\infty\}$ with $M : \underline{\underline{N}} + \{\infty\} \longrightarrow N + \{\infty\} + 1$ the difference function $M(n) = n-1$ if $n > 0$, $M(0) = \perp$ (where $1 = \{\perp\}$), $M(\infty) = \infty$. The universal property is




Here

$$M(\psi(q)) = \begin{cases} \infty & \text{if } \psi(q) = \infty \\ \psi(q) - 1 & \text{if } \psi(q) > 0,\ \psi(q) \neq 0 \\ \perp & \text{if } \psi(q) = 0. \end{cases}$$

But

$$(\psi + 1)\ (q) = \begin{cases} \psi(\Delta(q)) & \text{if } \Delta(q) \text{ is defined} \\ \perp & \text{if not.} \end{cases}$$

Thus $\psi(q) = 0$ if $\Delta(q)$ is undefined, while $\psi(q) = n$ if $\Delta^k(q)$ is defined for $1 \leq k < n$ but $\Delta^n(q)$ is undefined, and $\psi(q) = \infty$ if $\Delta^k(q)$ is always defined.

The functor of (3) is of the form $X_\Omega$ (set $\Omega_A = Y$) and so is bicontinuous.

## 4. Stacks and Queues

Given a set E of elements, we may (as do, e.g., Lehmann and Smyth [1981]) obtain the stack of elements from E by forming the least fixpoint of the functor $XQ = 1 + E \times Q$, which is the isomorphism

$$\mu : 1 + E \times S \longrightarrow S .$$

This defines stack functions by

$$\Lambda = \mu \cdot in_1 : 1 \longrightarrow S, \quad \text{which defines the empty stack, and}$$
$$push = \mu \cdot in_2 : E \times S \longrightarrow S.$$

From $\mu^{-1} : S \longrightarrow 1 + E \times S$, we obtain the partial functions

$$\alpha : S \longrightarrow 1 \qquad \text{defined only on the empty stack, and}$$
$$\beta : S \longrightarrow E \times S \qquad \text{defined only on nonempty stacks, which}$$

decomposes to yield the two stack-functions

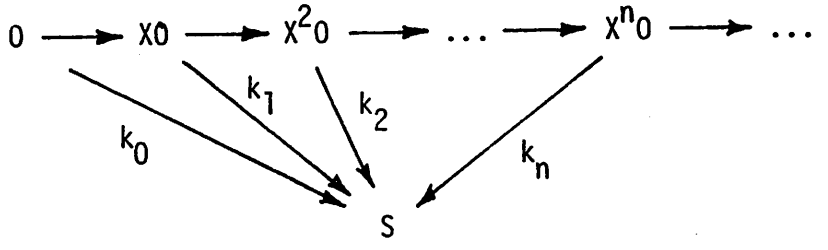
$$top = pr_1 \cdot \beta : S \longrightarrow E, \quad \text{and}$$
$$pop = pr_2 \cdot \beta : S \longrightarrow S.$$

In short, all the operations associated with the stack data type may be 'unpacked' from the isomorphism $\mu$, and so we may say that the least fixed point $\mu : 1 + E \times S \longrightarrow S$ is the data type.

It is then clear that the notion of stack is immediately available as a parametrized data type:

Definition: Let E be any object of a category $\mathcal{K}$ such that the functor $X_E Q = 1 + E \times Q$ is defined and co-continuous. Then the parametrized data type stack of ( ) is defined for E, and sends E to the least fixpoint $\mu_E : 1 + E \times S_E \longrightarrow S_E$ of $X_E$. Thus, for each suitable E in $\mathcal{K}$, stack of E is an object of the diagram category $\mathcal{K}^{\bullet \to \bullet}$ whose objects are single morphisms in $\mathcal{K}$.

Let us now look at the colimit construction for stack of E (abbreviating $X_E$ to X, and $S_E$ to S). S is defined by the colimit

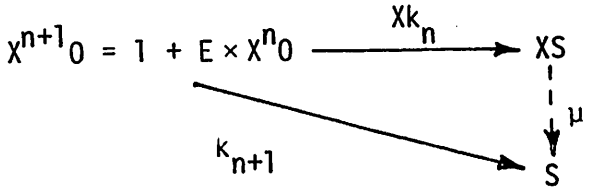



We note, by induction, that

$$X0 = 1 + E \times 0 \cong 1$$
$$X^2 0 = 1 + E \times 1 \cong 1 + E$$

and, generally, $X^n 0 \cong 1 + E + \ldots + E^{n-1}$, and we shall use the latter form for $X^n 0$.

Because the functor X preserves colimits, we have that $(X^{n+1}0 \mid n \geq 0)$ has colimit $(Xk_n : X^{n+1}0 \longrightarrow XS)$. Since we have the cocone $(X^{n+1}0 \xrightarrow{k_{n+1}} S)$, we obtain $\mu$ as the unique $XS \longrightarrow S$ such that



We now show that this construction enables us to define a queue as the FIFO version of the stack (which is LIFO). We will then contrast this specification with the equational specification approach.

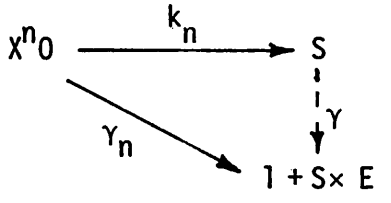
On the model of the definition of top and pop, we want to use the above definition of a stack to yield a queue simply by adding the definition for the partial functions (each with the nonempty stacks as domain of definition):

last: $S \rightarrow\!\!\!\!\!\!+\ E$ which returns the last element of a nonempty stack; and

front: $S \rightarrow\!\!\!\!\!\!+\ S$ which returns the stack obtained by deleting the last element from a nonempty stack.

We must do this using only the apparatus from the definition of $\mu$. Just as we obtained top and pop from $\mu^{-1} : S \longrightarrow 1 + E \times S$, so shall we now define a function $\gamma : S \longrightarrow 1 + S \times E$ from which we may obtain last and front. Note well that this construction goes through for any E which admits the definition of stack of E.

To define $\gamma : S \longrightarrow 1 + S \times E$, we define maps $\gamma_n : X^n 0 \longrightarrow 1 + S \times E$, and then obtain $\gamma$ as the unique solution of




The inductive definition of $\gamma_n$ proceeds as follows:

$$\gamma_0 = ! : 0 \longrightarrow 1 + S \times E$$
$$\gamma_1 = in_1 : 1 \longrightarrow 1 + S \times E.$$

Then, for $n \geq 1$, we motivate the definition of

$$\gamma_{n+1} : X^{n+1}0 = 1 + E \times X^n 0 \longrightarrow 1 + S \times E$$

by the set-theoretic case (noting that the E in $S \times E$ and $E \times X^n 0$ are 'at opposite ends'). Since

$$X^{n+1}0 = 1 + E \times (1 + E \times X^{n-1}0) \cong 1 + E \times 1 + E \times E \times X^{n-1}0$$

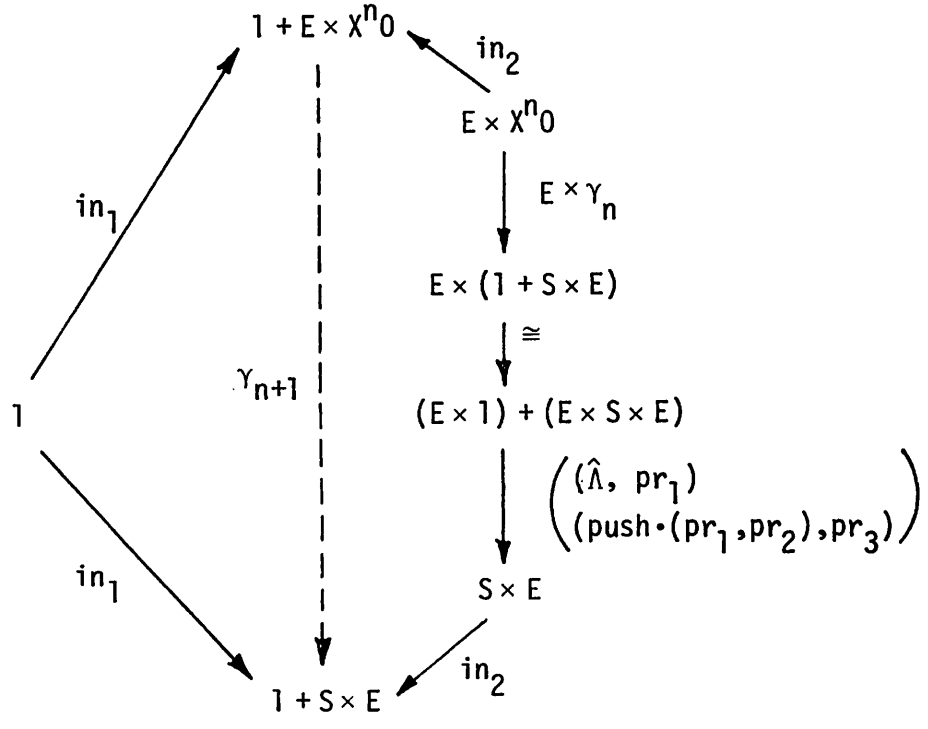
there are three situations for an element w of $X^{n+1}0$ in Set:

$$w = \Lambda \implies \gamma_{n+1} w = \Lambda \qquad \text{(taking } 1 = \{\Lambda\} \text{ in Set)}$$

$$w = (e, \Lambda) \implies \gamma_{n+1} w = (\Lambda, e)$$

$$w = (e,s) \text{ with } s \neq \Lambda \implies \gamma_{n+1}(w) = (e \cdot \mathrm{front}(s), \mathrm{last}(s)).$$

Given $\gamma_0$ and $\gamma_1$ as above, we use this motivation to define $\gamma_{n+1}$ $(n \geq 1)$ in the general setting by the diagram:




where $\hat{\Lambda} = \Lambda \cdot ! : E \times 1 \longrightarrow 1 \longrightarrow S$, and we have assumed our category such that products distribute over coproducts.

We have already mentioned the approach (let us call it ES) to the abstract specification of data types which uses an equational specification of many-sorted algebras. In this approach, a queue is defined by two sorts S and E, together with operators

$$\begin{aligned} \Lambda &: 1 \longrightarrow S \\ \text{push} &: E \times S \longrightarrow S \\ \text{front} &: S \longrightarrow S + \{\text{error}\} \\ \text{last} &: S \longrightarrow E + \{\text{error}\} \end{aligned}$$

subject to the equations (using the obvious obbreviations p, f and $\ell$):

$$\begin{aligned} f(\Lambda) &= \text{error} & \ell(\Lambda) &= \text{error} \\ f(p(e,\Lambda)) &= \Lambda & \ell(p(e,\Lambda)) &= e \\ f(p(e_1, p(e_2,s)) &= p(e_1, f(p(e_2,s))) & & \\ & & \ell(p(e_1, p(e_2,s))) &= \ell(p(e_2,s)). \end{aligned}$$

Confronted with these equations, the ES approach then constructs the initial algebra which corresponds to them. Given the nature of the functions involved, the construction is elaborate. By its nature, it does not make it at all obvious that the resultant data structure is indeed a "stack with FIFO retrieval".

However, the reader will immediately see that the intuition that led the ES theorist to write down the equations -- and his job is just begun -- is what led us to write down the inductive definition of $\gamma$ on the stack -- and our job is already completed. Specifically, the three pairs of equations correspond to the three pieces of

$$X^{n+1}0 = 1 + E \times (1 + E \times X^{n-1}0) \cong 1 + E \times 1 + E \times E \times X^{n-1}0$$

with typical elements $\Lambda$, $(e,\Lambda)$ and $(e_1, e_2, s)$.

## 4. Remarks by way of Conclusion

The main contribution of this paper is the systematic introduction of greatest fixpoints into the setting proposed by Lehmann and Smyth [1981] (but without emphasis on categories of ordered sets). Why have these greatest fixpoints not received attention previously? They were always available as the dual theory to that for least fixpoints. We offer two possible explanations.

The first evolves from a well-known method of assigning semantics to a recursive specification of a partial function $D \longrightarrow D$ for some set D. Regard the set $\underline{\underline{Pfn}}(D,D)$ of all such functions as an $\omega$-complete poset with the extension ordering and with least element the everywhere undefined function $\perp$. Usually, the specification

$$f ::= \psi(f)$$

is such that $\psi : \underline{\underline{Pfn}}(D,D) \longrightarrow \underline{\underline{Pfn}}(D,D)$ is continuous, and so by the theorem of Kleene [1952], it has the supremum of

$$\perp \leq \psi(\perp) \leq \psi^2(\perp) \leq \ldots$$

for least fixpoint (the desired semantics in most cases). As Lehmann and Smyth point out, this is a special case of the functorial least fixpoint. The details are well-known and take the form

| special case | general case |
|---|---|
| partially-ordered set | category |
| monotone map | functor |
| continuous map | co-continuous functor |
| least element | initial object |
| equality | isomorphism. |

The greatest fixpoint of $\psi$: $\underline{\underline{Pfn}}(D,D) \longrightarrow \underline{\underline{Pfn}}(D,D)$ need not exist because $\underline{\underline{Pfn}}(D,D)$ does not have a greatest element. We suggest, then, that one reason greatest fixpoints have been ignored is that people sought to generalize exclusively from recursive specifications of functions to recursive specifications of data types.

A second possible reason surfaced in conversation with Gordon Plotkin in June 1982. Apparently he and Smyth had considered the greatest fixpoint construction in $\underline{\underline{Set}}$ but had abandoned it because of an inability to deal with incomplete specification. For example, in the set $A^* + A^\infty$ arising from the greatest fixpoint of $XQ = (A \times Q) + B$ (with B a 1-element set), there are limit projections representing an infinite list as the sequence of its finite sublists, but there is no actual list of form

$$a_1 a_2 \cdots a_n \perp$$

where $\perp$ is an 'as yet undetermined' list. We leave it to the reader to judge if this objection is countered by the results of Section 3.3. And we certainly concede that there are enough least fixpoints of functors on categories of domains to produce the principal carrier of all data types of interest. Our advocacy of the greatest fixpoint construction is based on its universal property 3(2). It is hard to imagine how the observability map of 3(4) could arise using the universal property of a least fixpoint. Indeed, it is sometimes natural to use both universal properties together. Thus, the usual iterate $f^\dagger : A \longrightarrow B$ of $f : A \longrightarrow A + B$ arises as the composition

$$f^\dagger : A \xrightarrow{\ g\ } A^*B + A^\infty \xrightarrow{\ p\ } A^*B \xrightarrow{\ \underline{last}\ } B$$

where g is the historical iterate of Section 3 arising from the universal property of a greatest fixpoint, p is the partial function $p(w) = w$ with

domain $A^*B$ and $\underline{last} : A^*B \longrightarrow B$ maps $a_1 \ldots a_n b$ to b. But $\underline{last}$ arises from the universal property 3(1) of the least fixpoint of 3(5):

$$\begin{array}{ccc} (A\times A^*B)+B & \xrightarrow{\mu} & A^*B \\ \Big\downarrow{\scriptstyle (id\times \underline{last})+id} & & \Big\downarrow{\scriptstyle \underline{last}} \\ (A\times B)+B & \xrightarrow{h} & B \end{array}$$

where $h(a,b) = b$, $h(b) = b$.

Such examples suggest that the mathematical theory of data types will best be served by the explicit recognition of the greatest fixpoint, and by the freeing of the study of fixpoints from any necessary dependence on ordered objects.

## References

J. Adámek and V. Koubek: Least fixed point of a functor, J. Comp. Syst. Sci. 19 (1979) 163-178.

J. Adámek and V. Trnková: Varietors and machines, Technical Report 78-6 (1978), Computer and Information Science Dept., Univ. of Massachusetts at Amherst.

M.A. Arbib and E.G. Manes: Machines in a category: an expository introduction, SIAM Review 16 (1974) 163-192.

M.A. Arbib and E.G. Manes: Arrows, Structures and Functors, Academic Press (1975).

M.A. Arbib and E.G. Manes: Partially-additive categories and flow-diagram semantics, J. Algebra 62 (1980a) 203-227.

M.A. Arbib and E.G. Manes: The greatest fixpoint approach to data types, Proc. 3rd Workshop Meeting on Categorical and Algebraic Methods in Computer Science and System Theory, Dortmund, West Germany, Nov. 3-7, 1980b.

M. Barr: Coequalizers and free triples, Math. Zeit. 116 (1970) 307-322.

J.A. Goguen, J.W. Thatcher, E.G. Wagner and J.B. Wright: Abstract data types as initial algebras and correctness of data representations, Proc. Conf. on Computer Graphics, Pattern Recognition and Data Structures, May 1975.

J.V. Guttag: Abstract data types and the development of data structures, Communications of the ACM 20 (1977) 396-404.

S.C. Kleene: Introduction to Metamathematics, Van Nostrand (1952).

V. Koubek: Set functors, Comm. Math. Univ. Carolinae 12 (1971) 175-195.

D.J. Lehmann and M.B. Smyth: Algebraic specification of data types: a synthetic approach, Math. Systems Theory 14 (1981) 97-139.

B. Liskov and S. Zilles: Programming with abstract data types, Proc. ACM SIGPLAN Conf. Very High Level Languages, SIGPLAN Notices 9 (1974) 50-60.

D.S. Scott: Continuous lattices, in Toposes, Algebraic Geometry and Logic, Springer Lecture Notes in Mathematics 274 (1977) 97-136.

M.B. Smyth: Category-theoretic solution of recursive domain equations, Univ. of Warwick Theory of Computation Report No. 14 (1976) 12 pp.

J.W. Thatcher, E.G. Wagner and J.B. Wright: Data type specification, parametrization and the power of specification techniques, ACM Trans. Prog. Lang. and Systems (in press).