BULL. AUSTRAL. MATH. SOC. 18D99, 68P05, 68Q65
VOL. 40 (1989) [79–82]

DATA TYPES IN DISTRIBUTIVE CATEGORIES

R.F.C. WALTERS

The purpose of this note is to describe some of the standard data types of computer science in the language of distributive categories. We believe that in this way we have achieved a simplification and a formal clarification of the specification of these data types.

0. INTRODUCTION

A category is *distributive* if it admits finite products and infinite coproducts, and the finite products distribute over the coproducts. Clearly the category **Sets** of sets and functions has this property. In the following, let $\mathbf{C}$ be a distributive category. (Most of the definitions in this note require only finite coproducts, and in fact the product need only be a tensor product.)

1. STACKS

DEFINITION: If X is an object of $\mathbf{C}$, then a data type *stack of* X is an object S of $\mathbf{C}$ together with two arrows

$$push : 1 + XS \to S$$
$$pop : S \to 1 + XS$$

such that

$$push \circ pop = 1_S$$
$$pop \circ push = 1_{1+XS}.$$

In short, S is isomorphic to $1 + XS$. The arrows *push* and *pop* are inverse.

Let us look at this example in **Sets** to see the relation between this definition and the definition in [**1**, p.305]. Note that 1 is a one element set (whose element may be denoted $*$ or *error*). Further, an arrow $1 + XS \to S$ is a pair of arrows $1 \to S$,

Received 20 September 1988
The author would like to thank F.W. Lawvere for many helpful suggestions he gave during his visit to Sydney in 1988 supported by the Australian Research Grants Scheme, and he would like to thank the members of his course on categories and computer science - in particular Phil Lavers, Gordon Monro and Bill Unger - for their encouragement.

$XS \to S$. In [1] the first of these is called *new*, and the second *push*. Now our equation $push \circ pop = 1_S$ implies that if $pop(s) = error$ then $s = new$, and $pop \circ push = 1_{1+XS}$ implies that $pop(new) = error$. Hence *pop* restricted to non-new stacks lands in XS. An arrow into a product is a pair of arrows, and so *pop*, restricted to non-new stacks, can be regarded as a pair of arrows, $pop: S \to S$, and $read: S \to X$ (this is the notation of [1]). In our description *empty* (of [1]) does not occur as basic data, but it is definable as $empty = (true + false) \circ pop$. The equations of [1] are all consequences of the fact that our *pop* and *push* are inverse. The notion of type stack in [1] is however weaker than ours since there is no requirement in [1] that *push* reverse the effect of *pop*.

The apparently illegitimate calculation

$$\begin{aligned} S &= 1 + XS \\ S - XS &= 1 \\ (1 - X)S &= 1 \\ S &= \frac{1}{(1 - X)} \\ S &= 1 + X + X^2 + X^3 + \cdots \end{aligned}$$

suggests the standard example of type stack of X.

2. Queues

Definition: If X is an object of $\mathbf{C}$, then data type *queue of* X is an object Q of $\mathbf{C}$ together with two arrows

$$\begin{aligned} push &: 1 + XQ \to Q \\ pop &: Q \to 1 + QX \end{aligned}$$

such that

$$\begin{array}{ccc} 1 + XQ & \xrightarrow{push} & Q \\ {\scriptstyle 1+X pop}\downarrow & & \| \\ 1 + X(1 + QX) & & Q \\ \| & & \downarrow{\scriptstyle pop} \\ 1 + (1 + XQ)X & \xrightarrow[1+push X]{} & 1 + QX \end{array}$$

In **Sets**, by a similar analysis to that above, a type queue in our sense yields a type queue in the sense of [1, p.306]. In fact, the only difference between our notion and that of [1] is that their *pop* and *read* are defined on empty queues (resulting, respectively, in a queue and a value).

3. Binary Trees

DEFINITION: If X is an object of $\mathbf{C}$, then a data type *binary tree of* X is an object B together with two arrows

$$make : 1 + BXB \to B$$
$$break : B \to 1 + BXB$$

such that

$$break \circ make = 1_{1+BXB}$$
$$make \circ break = 1_B.$$

As in our description of type stack, in **Sets** our arrow *make* corresponds to two functions of [1], namely *new* and *make*. Further, *break* corresponds to three functions of [1], *left*, *data* and *right*. Our equations imply the equations of [1]. Again the notion of type binary tree in [1] is weaker than ours since it does not insist that *make* reverse the effect of *break*.

The apparently illegitimate calculation

$$BXB + 1 = B$$
$$XB^2 - B + 1 = 0$$
$$B = \frac{1 - \sqrt{1 - 4X^2}}{2X}$$
$$B = \sum_{n=0}^{\infty} \frac{1}{n+1} \binom{2n}{n} X^n$$

suggests the standard example of binary tree of X.

4. Arrays of fixed length

DEFINITION: The data type *array of* X *of length* n is X^n.

The data type array has no particular specified operations. However there are many operations which are definable from the properties of a distributive category.

For example, in a distributive category X^n is the exponential object X to the power $1 + 1 + \cdots + 1$, because of the following sequence of bijections

$$\frac{\frac{\frac{Y \to X^n}{(Y \to X)_{i \in N}}}{Y + Y + \cdots + Y \to X}}{(1 + 1 + \cdots + 1)Y \to X.}$$

This means that X^n has a *fetch* arrow, namely evaluation. The *store* arrow $X^n \times n \times X \to X^n$ arises in an obvious way from n arrows $X^{n+1} \to X^n$, defined from projections.

5. Lists with Pointer

DEFINITION: If X is an object of $\mathbf{C}$, then a data type *list of X with pointer* is an object L of the form SXS where S is stack. This is suggested by the standard example in **Sets**

$$L = \sum_{n=1}^{\infty} n \times X^n = \left(\frac{X}{(1-X)^2} \right)$$

in which an element is a list of length n for some n, with an address out of n.

The specified operations are precisely those of the two stacks. Again there are many further operations definable in a distributive category for a type list with pointer and some are listed below. The arrow ! is the unique arrow with codomain 1.

Overwrite the contents of X by the constant $\alpha: 1 \to X$

$$S\,X\,S \xrightarrow{S\alpha\,X\,S} S\,X\,X\,S \xrightarrow{S\,X!S} S\,X\,S,$$

Insert constant α in X shifting the contents to the right

$$S\,X\,S \xrightarrow{S\alpha X\,S} S\,X\,X\,S \xrightarrow{S\,push} S\,X\,S,$$

Read the contents of X

$$\text{projection: } S\,X\,S \to X,$$

Move the pointer one to the right

$$S\,X\,S \xrightarrow{S\,X\,pop} S\,X\,(1 + X\,S) = S\,X + S\,X\,X\,S \xrightarrow{!+push\,X\,S} 1 + S\,X\,S.$$

References

[1] A. Berztiss and S. Thatte, 'Specification and implementation of abstract data types', *Adv. Comput.* **22** (1983), 295–353..

Pure Mathematics Department
University of Sydney NSW 2006
Australia