Journal of Pure and Applied Algebra 66 (1990) 1–29
North-Holland

LIST-ARITHMETIC DISTRIBUTIVE CATEGORIES: LOCOI

J.R.B. COCKETT *
Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301, USA

Communicated by M. Barr
Received 12 June 1989
Revised 6 August 1989

A finitely complete category with stable disjoint coproducts and a parameterized list construction is called a *locos*. The paper proves that the property of being a locos is local in the sense of being inherited by slice categories. This is proven by establishing two important properties of the list construction in this setting.

The first of these is that lists can be characterized as objects which satisfy a domain equation and have their tail maps *contractions*. A contraction is an endomorphism which, when it is applied frequently enough, becomes fixed. It is a central technical notion in this development. As this characterization only uses the number arithmetic of the setting, it provides a powerful tool for establishing the existence of lists.

The second result is that list construction preserves and creates connected limits. Because lists satisfy a domain equation they are models of a certain type of sketch. Models of such sketches are preserved and created by connected limits. However, it is the fact that the contractions are also preserved by these limits which is crucial. This observation immediately gives the localness of list construction in a locos and is fundamental to many of the other properties of list construction.

1. Introduction

The purpose of this paper is to develop the basic properties of list construction over a distributive category. A *distributive category* is a finitely complete category with finite stable disjoint coproducts.[1] As the setting over which lists are constructed greatly influences the result, it is worth discussing the setting chosen.

Distributive categories are of interest in their own right. They have many nice mathematical properties: for example, they are local and the initial distributive category is the category of finite sets. Furthermore they give a very natural setting for the discussion of computational properties. In particular, they are about the

* Present address: School of Mathematics, Physics, Computing and Electronics, Macquarie University, NSW 2109, Australia.

[1] These categories were named by F.W. Lawvere. Confusingly, this term has also been used to describe categories with binary coproducts and finite products which distribute over the coproduct. To avoid this confusion we suggest the name *predistributive* for the latter concept.

0022-4049/90/$03.50 © 1990 — Elsevier Science Publishers B.V. (North-Holland)

simplest setting in which the recognition of subobjects (that is 'recursive subsets') can be sensibly discussed.

All the constructs assumed in distributive categories have a very natural computational interpretation. Products are records while coproducts are variant records. While it is harder to justify equalization, it certainly has a clear computational meaning and allows the representation of objects as constrained subobjects of structured objects (for example the rational numbers, $\mathbb{Q}$, are an equalized subobject of $\mathbb{Z} \times \mathbb{N}$). Finally, the stability and disjointness are the properties which allow the recognizable subobjects to be identified as those which have complements.

Unfortunately, distributive categories do not have any data structures and this limits their applicability. However, when a parameterized list constructor is added this changes. Suddenly all the free constructions are present. Such a category is called a *locos*. These categories are of particular interest both in initial algebra semantics for programming languages [8], and more generally in theoretical computer science. They give a very constructive setting for a mathematics based on list construction rather than power set construction.[2] The elements of the initial locos[3] can be directly interpreted as programs[4] and can be rewritten mechanically to a canonical form much as terms in initial Cartesian closed categories can be [12].

A category with a parameterized natural number object is called a (*number-*) *arithmetic category*, while a category with a parameterized list constructor is called a *list-arithmetic category*. In order to support parameterization the category has to have products. Even when a category is not list-arithmetic it is possible that it has objects, $\mathrm{rec}(A, B)$, which have all the formal properties of a list with its parameter, $\mathrm{list}(A) \times B$. Such an object is called a *recursive* on A and B. Any R which is recursive on A and B must be an initial solution to the domain equation $B + A \times R \cong R$. Conversely, however, a solution to this domain equation need not be recursive. In fact, we shall call such solutions *semirecursives* and loosely refer to them as infinite objects.

In a distributive category the key structure is the (number-)arithmetic. Once this is in place to be list-arithmetic it suffices to have enough infinite objects. In fact, although this is beyond the scope of this paper, once these infinite objects are present, all free internally specified algebraic constructions are present (this is the 'fundamental theorem of data structures') and conversely such constructs will (usually) generate all infinite objects.

This paper develops two key results concerning list arithmetic over a distributive category. The first result is a characterization of recursive objects in terms of a (number-)arithmetic property and the **FLS**-sketchable property of being an infinite

[2] In fact, it might reasonably be argued that the type of mathematics determined by this setting is precisely what is traditionally called 'discrete mathematics'.

[3] Such exists, as a locos is **FL**-sketchable [17].

[4] The *charity* programming language, developed by the author and H.G. Chen, implements a *lazy* rewriting of locos elements following the work of [10].

object. This characterization is reminiscent of Freyd's characterization of the natural number object in a topos [7]. The characterization has the significant consequence, which is the second main result, that connected limits of recursives are recursive. An immediate consequence of this is that the list construction is local, in the sense of being inherited by the slice categories. Another immediate consequence, not discussed in this paper, is that locoi built from decidable objects have every object decidable [6].

This paper improves upon earlier work of the author [5, 6]. In the earlier work the setting is a list-arithmetic regular distributive category. Regularity provides existential quantification. Computationally this is hard to justify. While it is the case that the initial locos is regular (number arithmetic is very potent [14, 16]), this property vanishes when a map which is not recursively given is added. The addition of such a map in a computational setting arises whenever one lacks total information about its implementation (e.g., when a subroutine developed independently is used as a black box).

Regularity was certainly used in obtaining proofs in this earlier work. However, it turned out that it was only used insofar as it was guaranteed anyway by the number arithmetic. The removal of regularity has led to a much cleaner theory. Of course existential quantification had to be replaced by something, and the proofs below rely heavily on the properties of *contraction maps*.

Contraction maps are the subject of Section 3. A contraction map is an endomorphism of an object which, if done enough times, collapses the object to its fixed subobject. Furthermore, the number of times it is necessary to apply the endomorphism to ensure that the fixed subobject is reached must be *calculable* from the input. The generic example of a contraction is the operation of taking the tail of a list (**cdr** in Lisp). If you do this a number of times equal to the length of the list, then one will obtain the empty list. In fact any object with a contraction whose fixed subobject is recognizable has a unique *length* map which gives the least number of contractions necessary to move into the fixed subobject.

A semirecursive object always has the analogue of the tail map of a list. If this 'tail' map is a contraction, the object is said to be *prerecursive*. The first main result is that prerecursive objects are actually recursive.

Subobjects which are closed to a contraction and contain their preimage under contraction, are *separator* subobjects for the contraction. Separator subobjects are totally determined by their intersection with the fixed subobject of the contraction. This can be used to show that each semirecursive contains a prerecursive as a minimal semirecursive subobject. Thus, the existence of semirecursives assures the existence of prerecursives, which, in turn, provide recursives.

To show that semirecursives and prerecursives are stable under connected limits it suffices to check that they are stable under equalization and pulling back. As all objects in the slice category can be constructed by a pullback of objects in the mother category, and these objects certainly permit list construction, we may construct the list of an arbitrary object in the slice by a pullback of the constructions

in the mother category. This shows that locoi are local.

Two notations are used throughout this paper. The first is 'categorical notation' (or 'categorical combinators') using composition on the right (the way things are in diagrams), and the second is 'equational notation' which uses function application on the left and infix operators. One notation can be translated into the other very easily:

$$[(x,(y,z))\mapsto f(z,g(x,y))] : X\times(Y\times Z)\to W$$

is an *abstracted map* in equational notation and is the map

$$\langle p_1 . p_1, \langle p_0, p_1 . p_0\rangle . g\rangle . f : X\times(Y\times Z)\to W$$

in categorical combinators. The big advantage of the equational notation is that it hides projections. Strictly speaking, equational logic is concerned with the equality of abstracted maps and an equation should be written as

$$[(x,(y,z))\mapsto f(z,g(x,y))] = [(x',(y',z'))\mapsto h(x',z')].$$

However, as the abstracted maps must have the same domain it is sensible to rename the variables to make the variable bases the same (if they are not already) and write this as:

$$f(z,g(x,y)) =_{(x,(y,z))} h(x,z).$$

This convention nearly follows [12], although in that notation the structure of the domain is not explicitly provided; instead the set of variables are simply listed. Clearly for proofs the detailed structure of the product is irrelevant, however, as an alternative method for specifying maps, this detailed structure *is* important. In this paper we shall use structured domains. However, we shall omit the subscript of the equality altogether if the equality is true for any variable base which includes those variables in the expression.[5]

We also need a notation to express maps from a coproduct. For this we shall use the notation

$$f : X_0 + X_1 \to Y; \qquad x\mapsto \begin{bmatrix} b(x_0)\mapsto g_0(x_0) \\ b(x_1)\mapsto g_1(x_1)\end{bmatrix}(x)$$

or simply

$$f : X_0 + X_1 \to Y; \qquad \begin{cases} b(x_0)\mapsto g_0(x_0) \\ b(x_1)\mapsto g_1(x_1)\end{cases}$$

to denote the map which in combinator notation is

$$\langle g_0; g_1\rangle : X_0 + X_1 \to Y.$$

[5] This last convention should be credited to M. Barr who strongly objected to my scrupulous subscripting!

A more complex use of the coproduct in defining a map, f, is to let the map behave like one of a number of maps, f_0 and f_1, depending on the outcome of some other map g. Traditionally this is described by saying "$f(x)$ has value $f_0(x, g(x))$ if $g(x)$ meets the first condition, etc." We shall use the following notation to express such a map

$$f: X \to Y; \qquad x \mapsto \begin{bmatrix} b_0(y_0) \mapsto f_0(x, y_0) \\ b_1(y_1) \mapsto f_1(x, y_1) \end{bmatrix} (g(x))$$

where $g: X \to Y_0 + Y_1$ and $f_0: X \times Y_0 \to Y$, $f_1: X \times Y_1 \to Y$. This translates to the combinator expression:

$$\langle g, i \rangle . d_0 . \langle f_0; f_1 \rangle$$

where $d_0: (Y_0 + Y_1) \times X \to Y_0 \times X + Y_1 \times X$ is the distribution isomorphism.

2. Preliminaries

Let $\mathbf{X}$ be a category with products. By $\mathrm{Act}_A(\mathbf{X})$ shall be denoted the category of *actions* (or *dynamorphisms* in the terminology of Arbib and Manes [1]) of $\mathbf{X}$. This means that the objects of $\mathrm{Act}_A(\mathbf{X})$ are arrows

$$x: A \times X \to X$$

while the morphisms are commutative squares

$$\begin{array}{ccc} A \times X_1 & \xrightarrow{x_1} & X_1 \\ {\scriptstyle i_A \times f}\downarrow & & \downarrow{\scriptstyle f} \\ A \times X_2 & \xrightarrow{x_2} & X_2 \end{array}$$

One may regard the objects of this category as state-changing devices on an input alphabet A. The map which gives the state change gives an *action* of A on the states. The maps are then the natural notion of homomorphism for these devices.

There is an obvious underlying functor, $U_A: \mathrm{Act}_A(\mathbf{X}) \to \mathbf{X}$, which simply forgets the action. The category shall be said to have *enough recursive objects*, or simply, to be *recursive* in case for each object A this U_A has a left adjoint F_A. When the various adjunction formulae are unwound, this means that for every pair of objects A and B there is an object $\mathrm{rec}(A, B)$ with the two maps:

$$r_0^{A,B}: B \to \mathrm{rec}(A, B),$$

$$r_1^{A,B}: A \times \mathrm{rec}(A, B) \to \mathrm{rec}(A, B),$$

which have the universal property for actions. Indeed any pair of maps such as these which have the universal property for actions determine a *recursive object* on A and

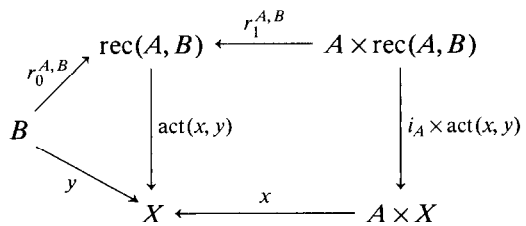
B. The universal property means that given any action

$$x : A \times X \to X$$

and a map

$$y : B \to X$$

there is a unique act(x, y) which makes the following diagram commute:



The property of having enough recursive objects can be expressed in various ways. The most common alternative ways are given in the following proposition:

Proposition 2.1. *For a category* **X** *with products the following are equivalent*:

(i) **X** *has enough recursive objects.*

(ii) *Each* $U_A : \mathrm{Act}_A(\mathbf{X}) \to \mathbf{X}$ *has a left adjoint.*

(iii) *Each endofunctor* $A \times -$ *has an associated free triple, whose category of algebras is* $\mathrm{Act}_A(\mathbf{X})$.

(iv) *For each endofunctor* $F = B + A \times \mathbf{X}$ *the category of actions* $\mathrm{Act}_F(\mathbf{X})$ *has an initial object* rec(A, B).

However, the form of the recursion generated by any of these specifications does not seem to have sufficient structure to give arithmetic.[6] Indeed, there seems to be no reason why the recursive objects, rec(A, B), should be at all well-behaved. What is lacking is the ability to do *parameterized* recursion from which primitive arithmetic arises.

The parameterization of a construct is only of interest in a setting in which exponentiation is not assumed. The parameterized version of a construction is, of course, already present when exponentiation is assumed. In weaker settings it is often necessary to strengthen constructs in this manner in order to obtain a sensible generalization. This is the case for list construction.

To obtain parameterized recursion it suffices to demand that the recursive objects in slice categories are closely related to those in the overlying category. It is certainly not obvious that the slice category of a category with enough recursive objects need itself have enough recursive objects. However, what is clear is that the slice category inherits many of the recursive objects.

[6] It is interesting to note that the initial models do have a well-behaved arithmetic [4].

Let **X** have products and let Y be an object of **X**, then we shall denote the overlying functor

$$V_Y : \mathbf{X}/Y \to \mathbf{X}; \qquad [F : X \to Y] \mapsto X,$$

and the right adjoint to this by

$$I_Y : \mathbf{X} \to \mathbf{X}/Y; \qquad X \mapsto [p_1 : X \times Y \to Y].$$

Lemma 2.2.

$$\mathrm{rec}(I_Y(A), I_Y(B)) = [\mathrm{act}(p_1^{A \times Y}, p_1^{B \times Y}) : \mathrm{rec}(A, B \times Y) \to Y].$$

Now there is a unique comparison map

$$\mathrm{act}(\langle \langle p_0, p_1 . p_0 \rangle . r_1, p_1 . p_1 \rangle, r_0 \times i_Y) : \mathrm{rec}(A, B \times Y) \to \mathrm{rec}(A, B) \times Y$$

and it is clear that demanding that this map is an isomorphism is equivalent to requiring that the functor I_Y preserves the recursion.

Definition 2.3. The recursion is said to be *parameterized* in case the functor, I_Y, preserves recursion in the above sense.

Of course this makes the *list constructor* extremely important in categories with parameterized recursion as the following series of isomorphisms shows

$$\mathrm{rec}(A, B) \to \mathrm{rec}(A, 1 \times B) \to \mathrm{rec}(A, 1) \times B \to \mathrm{list}(A) \times B$$

where it is clear that $\mathrm{list}(A)$ is $\mathrm{rec}(A, 1)$. The object B is now clearly identifiable as the *parameter* of the recursion. Furthermore, it is not hard to show that $\mathrm{list}(A)$ is the free monoid on A [14, 5].

The following notions are useful to distinguish the level of recursion which is present in a category:

Definition 2.4. A category **X** which has products is said to be:

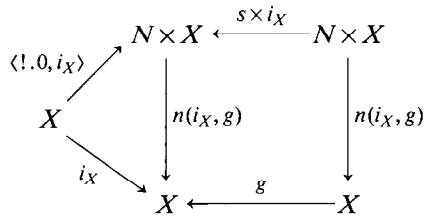
(i) *(number-)arithmetic* if for every B the recursive object $\mathrm{rec}(1, B)$ exists and is parameterized,

(ii) *list-arithmetic* if for every A and B the recursive object $\mathrm{rec}(A, B)$ exists and is parameterized.

Usually the 'number-' prefix will be dropped, so a category is arithmetic if it has a parameterized natural number object. An arithmetic category has all primitive recursive functions. The initial arithmetic category is the category of *formal primitive recursive functions* [4, 15]. While this category is not equivalent to the primitive recursive functions in **Sets**, in particular equality of maps is not determined by elements, all the basic arithmetic functions and their properties are still valid. These results are, for example, developed in [14–16] and will be assumed in

this paper.

A map which is frequently used is the solution to the following recursion scheme



We shall predominantely use the rather suggestive equational notation

$$n(i_X, g) : N \times X \to X; \qquad (n, x) \mapsto g^n(x)$$

of raising g to the power n for this map. All the parameterized number arithmetic maps can be expressed using this map, thus it is a convenient notation. It also suggests that certain identities should hold:

$$g^0(x) = x, \qquad g^{s(n)}(x) = g(g^n(x)), \qquad g^{n_1 + n_2}(x) = g^{n_1}(g^{n_2}(x)).$$

They are in fact valid identities: the first two are the defining properties of $g^n(x)$ and the third can be easily proved by using the uniqueness of the recursion scheme.

A useful technique for proving that two maps $f, g : N \times X \to Y$ are equal is to check that $f(0, x) = g(0, x)$ and to prove that if $f(n, x) = g(n, x)$, then $f(s(n), x) = g(s(n), x)$. This, of course, is a standard proof by induction. However, we have no guarantee that it will actually work in the settings we are considering.

However, if the setting is a *finitely complete* arithmetic category, then this argument is certainly valid. This can be seen quite easily by examining the equalizer of f and g. This subobject is closed to the $\langle s \times i_X \rangle$ action and contains the base of the induction. However, any subobject closed to this action and containing the base must be the whole object.

3. Contractions

A *contraction* is an endomorphism of an object which, if done enough times, will reach a steady state. Obviously, how many times the contraction should be performed to reach that steady state will be dependent on the input. To be a contraction, there must actually be a way of *calculating* this number from the input.

There are many natural examples of contractions. Once the primitive rewriting step is defined, a terminating rewriting system gives a contraction on the terms of the algebra. Also, when termination is determined by the state becoming fixed, the 'tick' of a Turing machine together with its input tape if it represents an algorithm, must be a contraction.

From now on we shall assume without explicit mention that all our categories are distributive and (number-)arithmetic.

Definition 3.1. An endomorphism, $f: X \to X$, is a *contraction* if there is a map $\beta: X \to \mathbb{N}$, called a *bounding map* of f with

$$f^{\beta(x)}(x) = f^{\beta(x)}(f(x)).$$

We shall refer to the equalizer of an endomorphism f with the identity as the *fixed subobject* of f. Let $c(x) := f^{\beta(x)}(x)$, then the following are useful observations concerning contractions:

Lemma 3.2. *If $f: X \to X$ is a contraction*:
(i) $f(x) = x$ *implies* $c(x) = x$,
(ii) $f(c(x)) = c(f(x)) = c(x)$,
(iii) $c(c(x)) = c(x)$.

The first two parts of this lemma show that the fixed subobject of f is the same as the image of c, and the third shows that c is an idempotent which collapses everything to the fixed subobject of f.

We wish to consider maps, from an object with a contraction to the natural numbers. In this discussion it is useful to have a name for the maps which are zero over the fixed subobject of the contraction: we shall call such maps *flows* over the contraction. This allows us to state the main result of the section as follows:

Theorem 3.3 (Contraction mapping theorem). *If $f: X \to X$ is a contraction with bounding map β and h is a flow over f, then there is a unique flow g over f such that $g(x) = g(f(x)) + h(x)$.*

Let $\varrho : \mathbb{N} \times X \to \mathbb{N}$ be defined by $\varrho(0, x) = 0$ and $\varrho(s(n), x) = \varrho(n, x) + h(f^n(x))$, then we define $g(x) = \varrho(\beta(x), x)$. We must show that this map has the desired properties. Towards this end we first establish some identities:

Lemma 3.4. *With ϱ, h, and f as defined above*:
(i) $\varrho(s(n), x) = \varrho(n, f(x)) + h(x)$,
(ii) $\varrho(\beta(x), x) = \varrho(\beta(x) + n, x)$.

Proof. (i) When $n = 0$, using both parts of the definition of ϱ, we have

$$\varrho(1, x) = \varrho(0, x) + h(x) = \varrho(0, f(x)) + h(x).$$

Assuming the lemma is true for n, we have that

$$\begin{aligned}\varrho(s(s(n)), x) &= \varrho(s(n), x) + h(f^{s(n)}(x)) \\ &= \varrho(n, f(x)) + h(x) + h(f^n(f(x))) \\ &= \varrho(s(n), f(x)) + h(x).\end{aligned}$$

(ii) First observe as $f(f^{\beta(x)+n}(x)) = f^n(f(c(x))) = f^{\beta(x)+n}(x)$ that $f^{\beta(x)+n}(x)$ is fixed by f and hence that $h(f^{\beta(x)+n}(x)) = 0$. When $n = 0$, the result is true. Now, assuming it is true for n, we have

$$\begin{aligned}\varrho(\beta(x) + s(n), x) &= \varrho(\beta(x) + n, x) + h(f^{\beta(x)+n}(x)) \\ &= \varrho(\beta(x), x). \qquad \square\end{aligned}$$

The theorem can now be proven:

Proof of Theorem 3.3. We have:

$$\begin{aligned}g(x) &= \varrho(\beta(x), x) \\ &= \varrho(\beta(x) + s(\beta(f(x))), x) \\ &= \varrho(s(\beta(x) + \beta(f(x))), x) \\ &= \varrho(\beta(f(x)) + \beta(x), f(x)) + h(x) \\ &= \varrho(\beta(f(x)), f(x)) + h(x) \\ &= g(f(x)) + h(x)\end{aligned}$$

and so g certainly satisfies the conditions of the theorem. It remains only to show that g is unique in this respect.

Assume that g' is a flow over f and that $g'(x) = g'(f(x)) + h(x)$. Then observe that $g'(f^n(x)) + \varrho(n, x) = g'(x)$ as

$$\begin{aligned}&g'(x) + \varrho(0, x) = g'(x), \\ &g'(f^{s(n)}(x)) + \varrho(s(n), x) = g'(f^{s(n)}(x)) + h(f^n(x)) + \varrho(n, x) \\ &\qquad\qquad = g'(f^n(x)) + \varrho(n, x).\end{aligned}$$

However, as g' is a flow, we have

$$g(x) = \varrho(\beta(x), x) = g'(f^{\beta(x)}(x)) + \varrho(\beta(x), x) = g'(x),$$

which gives the uniqueness. $\square$

The original proof I proposed for this theorem was a generalization of a proof in Goodstein's book [9]. That proof was used to establish the equality $x + (y \dot{-} x) = y + (x \dot{-} y)$. Roland, under Joyal's guidance [14], had shown how Goodstein's proof could be transferred into a Cartesian category (no equalizers) and my original proof was tailored to that setting. The proof presented, however, uses the inductive method of proving equality discussed in the previous section and is a simplification of my proof due to Michael Barr.

Lambek [11] among others, has applied this result in the expansion form. That is, the map $\langle s \times s \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ was used rather than $\langle \text{pred} \times \text{pred} \rangle$ to obtain uniqueness for maps with the same flow. The expansion and contraction form are

closely related. However, the contraction form better captures the generality of the situation.

The importance of this theorem is that it gives a way of constructing maps to $\mathbb{N}$ from objects with contractions. It is worth noting that an object with a contraction already must have a map to $\mathbb{N}$, namely the bounding map β. As we shall be using maps constructed from flows over contractions it is useful to have a name for them. We shall call the map constructed in the theorem the *ground potential* of h over f.

Definition 3.5. If $f: X \to X$ is an endomorphism:

(i) $g: X \to \mathbb{N}$ is a *potential* over f if $g(f(x)) \leq g(x)$, and $\delta_f[g](x) := g(x) \dot{-} g(f(x))$ is the *flow* of g over f (here $\dot{-}$ is subtraction truncated at zero),

(ii) $g: X \to \mathbb{N}$ is a *strict potential* over f if g is a potential and $g(x) = g(f(x))$ implies $g(f(x)) = g(f^2(x))$. In other words the zero subobject of the flow of g is closed to f. That is, $\delta_f[g](x) = 0$ implies $\delta_f[g](f(x)) = 0$.

A flow is analogous to a differential or difference, while a potential is analogous to an integral or solution of a difference equation. The results we shall develop below also have obvious analogies along these lines. In the same way that an integral is determined by boundary conditions we may introduce these into the contraction mapping theorem:

Corollary 3.6 (Contraction mapping theorem with boundary conditions). *If f is a contraction map, h is a flow for f, and $r: [f = i_X] \to \mathbb{N}$ is any map, then there is a unique map $g_r: X \to \mathbb{N}$, called the* potential *for h over f with* boundary condition *r, such that*:

$$g_r(p^{[f=i_X]}(y)) = r(y),$$

$$g_r(x) = g_r(f(x)) + h(x),$$

where $p^{[f=i_X]}: [f = i_X] \to X$ is the inclusion of the fixed subobject. □

It is clear that $g_r(x) = g(x) + r(c|_{f=i_x}(x))$ and that this definition satisfies all the required properties.

While it is beyond the scope of this article, an arbitrary map to $\mathbb{N}$ over a contraction can be totally described with the help of a negative flow $\delta_{-f}[g](x) := g(x) \dot{-} g(f(x))$. The map is then the potential of the positive flow less the ground potential of the negative flow.

Potentials over f are, roughly speaking, determined by their flow and boundary condition. This is only precisely correct, of course, when f is, or can be turned into, a contraction map.

Corollary 3.7. *If g is a potential for a contraction map f, then g is uniquely determined by being the potential for $\delta_f[g]$ with boundary condition $[x \mapsto g(p^{[i_X=f]}(x))]$.* □

This means that potentials are what we expect them to be over a contraction f. The interesting case arises when f is not a contraction. In this case it is often possible to modify f to make it a contraction. In particular, for strict potentials we have:

Proposition 3.8. *If $g : X \to \mathbb{N}$ is strict potential with respect to f, then*:

(i) *There is an endomorphism f_0 such that g is strict potential for f_0 and the fixed subobject of f_0 is precisely the zero subobject of $\delta_f[g]$.*

(ii) *If the fixed subobject of f is the zero subobject of $\delta_f[g]$, then f is a contraction map with bound g.*

Proof. (i) Define f_0 by:

$$f_0 : X \to X; \qquad x \mapsto \begin{bmatrix} 0 \mapsto x \\ s(n) \mapsto f(x) \end{bmatrix}(\delta_f[g](x)).$$

But now $g(f_0(x)) \leq g(x)$ as this is the case for both parts of the map. Furthermore, g remains strictly differentiable over f_0.

(ii) Define

$$g_0 : X \to \mathbb{N}; \qquad x \mapsto \begin{bmatrix} 0 \mapsto 0 \\ s(n) \mapsto g(x) \end{bmatrix}(\delta_f[g](x)),$$

then $g_0(x) \leq g(x)$ and $g_0(f(x)) \leq g_0(x)$.

But also $g_0(f(x)) \leq \mathrm{pred}(g_0(x))$ as while $\delta_f[g](x) = 0$ each side is zero, and otherwise we have:

$$g_0(f(x)) \leq g(f(x)) = g(x) \dot{-} \delta_f[g](x) = g_0(x) \dot{-} s(n) \leq \mathrm{pred}(g_0(x)).$$

Thus, $g_0(f^{g(x)}(x)) \leq g_0(x) \dot{-} g(x) = 0$ and hence $f(f^{g(x)}(x)) = f^{g(x)}(x)$ making $g(x)$ a bounding map. □

For a map $f : \mathbb{N} \to \mathbb{N}$ which has $f(x) \leq x$ we can set $g = i_{\mathbb{N}}$. Then $\delta_f[i_{\mathbb{N}}](x) = x \dot{-} f(x) = 0$ precisely when $f(x) = x$. So $i_{\mathbb{N}}$ is strictly differentiable with respect to f making f a contraction.

Corollary 3.9. *If $f : \mathbb{N} \to \mathbb{N}$ has $f(x) \leq x$, then f is a contraction.* □

This gives a ready source of contractions on $\mathbb{N}$!

It is reasonable to suppose that a contraction might have a least number of times that it needs to be applied to reach the fixed set. This intuition is basically correct:

Proposition 3.10. *If $g : X \to \mathbb{N}$ is a strict potential with respect to f, then there is a unique $l : X \to \mathbb{N}$ such that*:

(i) $l(f(x)) = \mathrm{pred}(l(x))$,

(ii) $\delta_f[g](f^n(x)) = 0$ *if and only if* $n \geq l(x)$.

Proof. Set l to be the ground potential for $h : X \to \mathbb{N}$; $x \mapsto \min(\delta_f[g](x), 1)$ over f_0, where f_0 is the contraction obtained from f by stopping f over the zero set of the differential. Then $l : X \to \mathbb{N}$ is a strict potential over f and clearly (i) holds.

Consider $R = \{(x, n) | g(f^n(x)) = 0\}$, then $\upsilon : R \to \mathbb{N}$; $(x, n) \mapsto n$ is a bounding map for the contraction $k : R \to R; (x, n) \mapsto (f_0(x), \text{pred}(n))$. Fixing this contraction for those (x, n) with $\delta_f[g](x) = 0$ gives a new contraction k_0.

Then $\delta_{k_0}[\upsilon](x, n) = \min(\delta_f[g](x), 1)$, but υ is not the ground potential so that for the ground potential, which is clearly $l(x)$, we have $l(x) \leq h(x)$, proving (ii). □

Definition 3.11. If $f : X \to X$ is a contraction, then a bounding map $\text{len} : X \to \mathbb{N}$ is a *length* map for the contraction if $\text{len}(f^n(x)) = 0$ if and only if $n \geq \text{len}(x)$.

Corollary 3.12. *The length map of a contraction exists if and only if the fixed subobject is complemented (or recognizable).*

Proof. If the fixed subobject is complemented, then the potential for the flow which is zero on the fixed subobject and one on its complement is strictly differentiable, and, in fact, the length map. Conversely, the zero subobject induced by a length map must be the fixed subobject; but this is clearly complemented by the non-zero subobject of the length map. □

There are a number of ways in which a 'contraction' can fail to have a length map. If the object is decidable, and there is a bounding map for the contraction, then, as the fixed subobject must then be recognizable, there will be a length map. For a decidable object, an interesting way in which an endomorphism, which should be a contraction, can fail to be a contraction is if the length map is 'too long', that is, it is a total function, but one which cannot be expressed internally. If there is a bounding map and yet no length map, then the object must be undecidable.

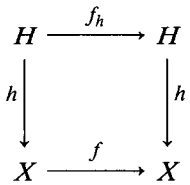
4. Separator subobjects

In order to be able to complete some of the proofs in the sequel it is necessary to develop some rather technical results concerning objects with actions. A simple action is an endomorphism, but we also ultimately have in mind the more general sorts of actions which give the action categories.

A subobject, $h : H \to X$ is *closed* to an endomorphism $f : X \to X$ if $h.f$ factors through h. Thus, the subobject h inherits the f-action. A subobject h *divides* the f-action if the pullback $f^{-1}(h)$ factors through h. It is obviously not the case that a closed subobject is necessarily a dividing subobject of the action. Nor is it the case that a dividing subobject must be closed to the action: consider $\mathbf{m} = \{n \in \mathbb{N} | n < m\}$; this divides the successor function but certainly is not closed to it.

We say that h is a *separator* of the f-action when it is both closed and divides the

action. That is $f(x)$ factors through h if and only if x does. Using pullbacks this says a subobject h is a separator if and only if



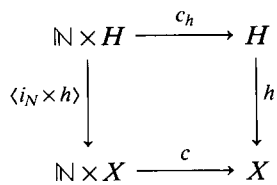
is a pullback. We shall use this diagram property extensively below. We start by showing that separators of $\langle i_X \times s \rangle$ have a rather special form:

Lemma 4.1. *If $h : H \to X \times \mathbb{N}$ is a separator for the action $\langle i_X \times s \rangle$, then $H \cong H_0 \times \mathbb{N}$ where $r_0 : H_0 = \{y \mid h(y) = (x, 0)\} \to H$.*

Proof. First there is an obvious map $H_0 \times \mathbb{N} \to H$ given by recursion on the action restricted to h, r_1. Furthermore this map is monic as the composite to $X \times \mathbb{N}$ is monic.

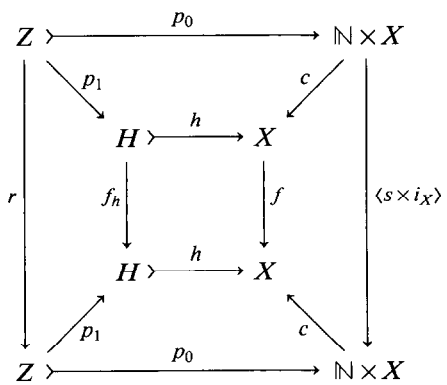
Now $\langle r_1; r_0 \rangle : H + H_0 \to H$ is an isomorphism, thus H is closed to the predecessor map $i_X \times \text{pred}$, so that $h = \langle h_0, h_1 \rangle$ can be contracted to h_1 which is in H_0. Thus h factors through $H_0 \times \mathbb{N}$. □

Proposition 4.2. *If $h : H \to X$ is a separator subobject for f, then letting $c_h : \mathbb{N} \times H \to H$; $(n, x) \mapsto f_n^n(x)$,*



is a pullback.

Proof. Consider the following commutative cube:



If Z is the pullback in question, then the big square is a pullback. However, by the above this means $Z \cong Z_0 \times \mathbb{N}$. However, Z_0 is clearly X. □

Corollary 4.3. *If* $h : H \to X$ *is a separator for a contraction, then letting* $c'_h : H \to [f_h = i_H]$; $x \mapsto f_h^{\beta(h(x))}(x)\big|_{f_h = i_H}$ *and* $c' : X \to [f = i_X]$; $X \mapsto f^{\beta(X)}(X)\big|_{f = i_X}$,

$$\begin{array}{ccc} H & \xrightarrow{c'_h} & [f_h = i_H] \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle h} \\ X & \xrightarrow{c'} & [f = i_X] \end{array}$$

is a pullback.

Proof. Compose the square of the proposition above with

$$\begin{array}{ccc} H & \xrightarrow{\langle h.\beta, i_H \rangle} & \mathbb{N} \times H \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle h} \\ X & \xrightarrow{\langle \beta, i_X \rangle} & \mathbb{N} \times X \end{array}$$

which is a pullback to obtain a new pullback whose horizontal maps factor through the fixed subobjects (which form a pullback square). □

This means that separator subobjects of contractions are determined by their intersection with the fixed subobject.

5. Recursive objects

An object is *recursive* on A and B if it satisfies the formal properties of $\mathrm{rec}(A, B)$. These properties may be expressed by saying that there is an initial B-parameterized A-action on the object. The purpose of this section is to characterize these objects.

Definition 5.1. An object R is *semirecursive* on A and B in case there are the following maps:

- $r_0 : B \to R$,
- $r_1 : A \times R \to R$,
- $\mathrm{split} : R \to B + A \times R$,

which satisfy the following conditions:

- $r_0 . \mathrm{split} = b_0$,
- $r_1 . \mathrm{split} = b_1$,
- $\mathrm{split} . \langle r_0; r_1 \rangle = i_R$,

that is, split is an isomorphism.

If in addition the map shrink $:=$ split $.\langle r_0; p_1\rangle$ is a contraction map with fixed subobject r_0, then the object is *prerecursive*.

A semirecursive is an 'infinite object' in the sense that it is a disjoint union of something with a parameterized version of itself. While our main concern shall be with prerecursives, semirecursives are of interest as not only do they provide a source of prerecursives, but also a convenient vehicle for studying their properties.

It is immediate from Corollary 3.12 that a prerecursive will have a length map. This length map satisfies some special properties:

Proposition 5.2. *Let R be a prerecursive*:

(i) $\mathrm{len}(r_0(b)) = 0$.

(ii) $\mathrm{len}(r_1(a, r)) = s(\mathrm{len}(r))$.

(iii) $\mathrm{len}(\mathrm{shrink}(r)) = \mathrm{pred}(\mathrm{len}(r))$.

(iv) $\mathrm{len}(\mathrm{shrink}^n(r)) = \mathrm{len}(r) \dot{-} n$.

(v) *The following square is a pullback*:

$$\begin{array}{ccc} A \times R & \xrightarrow{r_1} & R \\ {\scriptstyle p_1.\mathrm{len}}\downarrow & & \downarrow{\scriptstyle \mathrm{len}} \\ \mathbb{N} & \xrightarrow{s} & \mathbb{N} \end{array}$$

Proof. For (v): $\mathbb{N} \cong 1 + N$, and the preimage of 0 under len is r_0, and, as the complement of this is r_1 we obtain the pullback square above. □

If R is a semirecursive on A and B, there will be a prerecursive sitting inside R. This prerecursive is, in fact, *the* subsemirecursive which is generated by A and B.

Proposition 5.3. *If R is a semirecursive on A and B, then there is a subsemirecursive S on A and B which is prerecursive and, furthermore, it is the least subsemirecursive on A and B contained in R.*

To prove this we need to establish two things:

1. That there is a prerecursive subsemirecursive.
2. That this is the smallest subsemirecursive on A and B.

The second of these is given by the following result:

Lemma 5.4. *A prerecursive R on A and B has no proper subsemirecursive on A and B.*

Proof. Suppose $h : H \to R$ is a subsemirecursive on A and B, then I claim:

$$\begin{array}{ccc} H & \xrightarrow{\text{shrink}_H} & H \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle h} \\ R & \xrightarrow{\text{shrink}_R} & R \end{array}$$

is a pullback. To show this it suffices to show that this is so on the components of the semirecursives. That is, the squares obtained by prefixing the horizontal arrows with respectively r_0 and r_1 are both pullbacks. In the case of r_0 this is obvious. For r_1 the square becomes

$$\begin{array}{ccc} A \times H & \xrightarrow{p_1} & H \\ {\scriptstyle \langle i_A \times h \rangle}\downarrow & & \downarrow{\scriptstyle h} \\ A \times R & \xrightarrow{p_1} & R \end{array}$$

but this is also a pullback.

Therefore h is a separator subobject for the shrink action, and so is determined by its fixed subobject. This is B and thus h is an isomorphism. □

To obtain the proposition it remains to exhibit a subobject which is prerecursive on A and B.

Proof of Proposition 5.3. Consider $\upsilon : V \to \mathbb{N} \times R$ determined by $\{(n, r) | \text{shrink}^n(r) = r_0(b)\}$, then $\upsilon . p_0$ is a bounding map for $s = \langle pred \times \text{shrink} \rangle$ and for the stopped contraction $s_0 : V \to V$ which fixes the subobject $\{(n, r) | r = r_0(b)\}$ and otherwise is s. Let len_{s_0} be the length map for this contraction, then

$$\langle \text{len}_{s_0}, \upsilon . p_1 \rangle : V \to \mathbb{N} \times R$$

factors through V. As $\text{len}_{s_0}(\text{len}_{s_0}(n, r), r) = \text{len}_{s_0}(n, r)$ we have a retraction of V. The fixed subobject of this retraction is a subobject of V which under projection to R becomes a subobject of R as $\text{len}_{s_0}(n, r) = \text{len}_{s_0}(n, m)$.

It remains to show that this subobject is closed to the r_1 action. However, this is clear as acting by r_1 simply increments the length. □

The remainder of this section is devoted to proving that prerecursives are recursive objects:

Theorem 5.5. *In an arithmetic distributive category prerecursives are recursive.*

There are two stages in this proof. The first involves showing that for every action there is a comparison map of parameterized actions with the correct properties. We shall present two slightly different proofs of this. The second part involves estab-

lishing the uniqueness of this comparison map. For this one must show that any two comparison maps must be equal, and, given what we already know about prerecursives, this is quite simple:

Lemma 5.6. *If R is prerecursive on A and B and there is a B-parameterized A-action on X, that is, an $f: B \to X$ and a $g: A \times X \to X$, then there is at most one map $a: R \to X$ such that $r_0 . a = b$ and $\langle i_A \times a \rangle . g = r_1 . a$.*

Proof. Let a and a' be two such maps, then consider their equalizer. This subobject $x: R' \to R$ contains $r_0 : B \to R$ and is certainly closed to the A-action. However, it is also a separator subsemirecursive. This may be seen by considering the square:

$$\begin{array}{ccc} A \times R' & \xrightarrow{r_1'} & R' \\ {\scriptstyle \langle i_A \times x \rangle}\downarrow & & \downarrow{\scriptstyle x} \\ A \times R & \xrightarrow{r_1} & R \end{array}$$

If $y . x = \langle y_A, y' \rangle . r_1$, then $y . x . a = y . x . a'$. However, this means:

$$\begin{aligned} \langle y_A, y' \rangle . \langle i_A \times a \rangle . r_1 &= \langle y_A, y' \rangle . r_1' . a \\ &= \langle y_A, y' \rangle . r_1' . a' \\ &= \langle y_A, y' \rangle . \langle i_A \times a' \rangle . r_1 . \end{aligned}$$

Now as r_1 is monic we obtain $y' . a = y' . a'$ showing that $\langle y_A, y' \rangle$ and y factors through $\langle i_A \times x \rangle$. As x is monic, y must also factor through r_1'. However, this shows that the square is a pullback.

Finally as r_0 factors through x the pullback of this is this factorization. Putting these two pullbacks together then shows that R' is semirecursive on A and B and that x is a separator subsemirecursive. By Lemma 5.4 this means that x is an isomorphism and that $a = a'$. □

To complete the proof of Theorem 5.5 it is now only necessary to exhibit an action preserving map from an arbitrary prerecursive to an action. This we shall do in two slightly different ways. The first way (suggested by M. Barr) is to treat a prerecursive object as an array and to define the action by successively acting by the entries deepest in the prerecursive object. The second way involves using a pair of prerecursives as a two-way tape with a reading head. One can then adjoin an action to this so that whenever the tape shifts right the entry under the head is used to change state. The action preserving map is obtained by driving the tape left then right.

This second approach, which I originally followed, does not add much in proof theoretic terms. However, it is rather suggestive both from the point of view of how one might implement act(f, g) and over the question of simulating a Turing machine in this setting.

If a prerecursive object is to be viewed as an array, then we need a map which obtains the nth entry. Clearly, for this to make sense, it will be necessary to remain within the bounds of the array. To achieve this, $\mathbb{N} \times R$ can be split into two disjoint subobjects:

$$x_1 : X_1 = \{(n, r) | n < \text{len}(r)\} \to \mathbb{N} \times R,$$

$$x_2 : X_2 = \{(n, r) | n \geq \text{len}(r)\} \to \mathbb{N} \times R,$$

where X_1 is the object of pairs with indexes which are 'in bounds'.

For (n, r) in X_1 note that $\text{shrink}^{\text{len}(r) \dot{-} s(n)}(x)$ has length at least 1. This means that this map factors through r_1 by a map $\langle \text{entry, tail} \rangle : X_1 \to A \times X$

$$\begin{array}{ccc} X_1 & \xrightarrow{x_1} & \mathbb{N} \times X \\ {\scriptstyle \langle \text{entry, tail} \rangle}\downarrow & & \downarrow{\scriptstyle [(n,x) \mapsto \text{shrink}^{\text{len}(r) \dot{-} s(n)}(x)]} \\ A \times X & \xrightarrow{r_1} & X \end{array}$$

The map entry now associates with the index n the nth value pushed onto the prerecursive. This gives the (B-indexed) array representation of a prerecursive which was being sought.

Now suppose that $g : A \times C \to C$, then define

$$h : (\mathbb{N} \times R) \times C \to (\mathbb{N} \times X) \times C;$$

$$((n, r), c) \mapsto \begin{bmatrix} s(n) \mapsto ((s(n), r), g(\text{entry}(n, r), c)) \\ 0 \mapsto ((n, r), c) \end{bmatrix} (n \dot{-} \text{len}(r)).$$

Lemma 5.7. *For* $n \leq \text{len}(r)$

$$p_1(h^n((0, r_1(a, r)), c)) = p_1(h^n((0, r), c)).$$

Proof. Consider this in the form

$$p_1(h^{\min(n, \text{len}(r))}((0, r_1(a, r)), c)) = p_1(h^{\min(n, \text{len}(r))}((0, r), c)).$$

We shall show that the equalizer certainly contains $((0, r), c)$ and if it contains $((n, r), c)$, then it must contain $((s(n), r), c)$. This suffices as the equalizer then must be the whole of $(\mathbb{N} \times R) \times C$.

For $n = 0$ the result is obvious. Now suppose that the result holds for $((n, r), c)$ and consider $((s(n), r), c)$. There are two cases $s(n) = \text{len}(r)$ and $s(n) < \text{len}(r)$. The first case lies in x_2 and so the result is immediate. All the difficulty lies in showing that the result holds for $s(n) < \text{len}(r)$, but here we have

$$p_1(h^n((0, r_1(a, r)), c)) = p_1(h^n((0, r), c)) = c'$$

so that

$$p_1(h^{s(n)}((0, r_1(a, r)), c)) = p_1(h((n, r_1(a, r)), c'))$$

but

$$h(n, r_1(a, r), c') = (n, r_1(a, r), g(\text{entry}(n, r_1(a, r)), c')).$$

Now

$$\begin{aligned}\text{entry}(n, r_1(a, r)) &= p_1(\text{shrink}^{\text{len}(r_1(a, r)) \dot{-} s(n)}(r_1(a, r))) \\ &= p_1(\text{shrink}^{s(\text{len}(r)) \dot{-} s(n)}(r_1(a, r))) \\ &= p_1(\text{shrink}^{\text{len}(r) \dot{-} n}(r_1(a, r))).\end{aligned}$$

However, $s(n) < \text{len}(r)$ so that $\text{len}(r) = s(n) + s(m)$ which means

$$\text{len}(r) \dot{-} n = s(s(m)) = s(\text{len}(r) \dot{-} s(n))$$

allowing us to continue the simplification:

$$\begin{aligned}&= p_1(\text{shrink}^{\text{len}(r) \dot{-} s(n)}(\text{shrink}((r_1(a, r)))) \\ &= p_1(\text{shrink}^{\text{len}(r) \dot{-} s(n)}(r)) \\ &= \text{entry}(n, r).\end{aligned}$$

Substituting this back in the original equation gives

$$h((n, r_1(a, r)), c') = ((n, r_1(a, r)), g(\text{entry}(n, x), c'))$$

so that

$$p_1(h^{\min(n, \text{len}(r))}((0, r_1(a, r)), c)) = p_1(h^{\min(n, \text{len}(r))}((0, r), c)). \qquad \square$$

Now given $f : B \to C$ and $g : A \times C \to C$ define the following map:

$$a : X \to C; \qquad x \mapsto p_1(h^{\text{len}(r)}((0, r), f(c(r)))).$$

Lemma 5.8. *$b = r_0 . a$ and $\langle i_A \times a \rangle . g = r_1 . a$.*

Proof.

$$\begin{aligned}&p_1(h^{\text{len}(r_0(b))}((0, r_0(b)), f(c(r_0(b))))) \\ &\quad = p_1((0, r_0(b)), f(b)) = f(b), \\ &p_1(h^{\text{len}(r_1(a, r))}((0, r_0(r)), f(c(r)))) \\ &\quad = p_1(h(h^{\text{len}(r)}((0, r_1(a, r)), f(c(r))))) \\ &\quad = p_1(h((\text{len}(r), r_1(a, r)), p_1(h^{\text{len}(x)}((0, r), f(c(x)))))) \\ &\quad = g(\text{entry}((s(\text{len}(r)), r_1(a, x)), p_1(h^{\text{len}(r)}((0, r), f(c(r)))))).\end{aligned} \qquad \square$$

This completes the first method of proving the result.

In order to be able to view a pair of prerecursives as a two-way tape we need a number of *shift* maps. Let R be prerecursive on A and B. Given any $g : A \times C \to C$, an action of A on C, we may define:

$$\text{right} : R \times R \to R \times R; \qquad \begin{cases} (r_0(b), x) \mapsto (r_0(b), x) \\ (r_1(a, x), y) \mapsto (x, r_1(a, y)) \end{cases}$$

$$\text{left} : R \times R \to R \times R; \qquad \begin{cases} (x, r_0(b)) \mapsto (x, r_0(b)) \\ (x, r_1(a, y)) \mapsto (r_1(a, x), y) \end{cases}$$

$$\text{right}_g : (R \times R) \times C \to (R \times R) \times C; \qquad \begin{cases} ((r_0(b), x), c) \mapsto ((r_0(b), x), c) \\ ((r_1(a, x), y), c) \mapsto ((x, r_1(a, y)), g(a, c)) \end{cases}$$

$$\text{left}_g : (R \times R) \times C \to (R \times R) \times C; \qquad \begin{cases} ((x, r_0(b)), c) \mapsto ((x, r_0(b)), c) \\ ((x, r_1(a, y)), c) \mapsto ((r_1(a, x), y), c) \end{cases}$$

The idea of the maps left and right is to simulate the shifting of the position of a reading head on a (two-way) tape. The maps left_g and right_g are modifications of this which, on left shifting simply move the head, but on right shifting combine the head movement with an action on C.

For these maps we have the following identities:

Proposition 5.9.

(i) $\text{right}(x, y) = \text{left}(y, x)$.
(ii) $\text{right}_g . p_0 = p_0 . \text{right}$ *and* $\text{left}_g . p_0 = p_0 . \text{left}$.
(iii) $(x, y) = \text{right}(\text{left}(x, y))$ *if* $\text{len}(y) > 0$.
(iv) $p_0(\text{right}^n(x, y)) = \text{shrink}^n(x)$.
(v) $\text{len}(x) + \text{len}(y) = \text{len}(p_0(\text{right}(x, y)))) + \text{len}(p_1(\text{right}(x, y))$.
(vi) $\text{right}^n(\text{left}^n(r_0(b), y)) = (r_0(b), y)$.
(vii) $(x, y) = \text{right}^n(\text{left}^n(x, y))$ *if* $\text{len}(y) \leq n$.
(viii) $(r_0(b), y) = \text{right}^n(\text{left}^n(r_0(b), y))$.

All these are straightforward except possibly (vi) and (vii). For these we have:

Proof. (of (vi) and (vii)). For (vi) consider this in the form:

$$\text{right}^{\min(n, \text{len}(y))}(\text{left}^{\min(n, \text{len}(y))}(x, y)).$$

For $n = 0$ the result is true (that is $(0, (x, y))$ is in the equalizer of this map with the identity). Consider the case for $(s(n), (x, y))$:

If $n \geq \text{len}(y)$, then $\min(s(n), \text{len}(y)) = \min(n, \text{len}(y))$, so that if $(n, (x, y))$ is in the equalizer, so will be $(s(n), (x, y))$. It remains to check the case when $n < \text{len}(y)$. For this we have, as $\text{len}(y) > 0$, that $y = r_1(a, y')$. Furthermore, as

$$\min(s(n), \text{len}(r_1(a, y'))) = \min(s(n), s(\text{len}(y'))) = s(\min(n, \text{len}(y')))$$

we have:

$$\text{right}^{\min(s(n), \text{len}(r_1(a, y')))}(\text{left}^{\min(s(n), \text{len}(r_1(a, y')))}(x, r_1(a, y'))$$

$$= \text{right}^{\min(n, \text{len}(y'))}(\text{right}(\text{left}(\text{left}^{\min(n, \text{len}(y'))}(x, r_1(a, y')))))$$

$$\text{right}^{\min(n, \text{len}(y'))}(\text{left}^{\min(n, \text{len}(y'))}(x, r_1(a, y')))$$

where the middle shifts disappear as the right tape is not empty. Finally

$$\min(n, \mathrm{len}(y')) = \min(n, \mathrm{len}(r_1(a, y')))$$

as $n < \mathrm{len}(r_1(a, y')) = \mathrm{len}(y)$. Thus, the equalizer is closed to the s-action and therefore the whole domain.

Now for (vii) we have

$$\mathrm{right}^n(\mathrm{left}^n(r_0(b), y))$$
$$= \mathrm{right}^{n \dot{-} \mathrm{len}(y)}(\mathrm{right}^{\min(n, \mathrm{len}(y))}(\mathrm{left}^{n \dot{-} \mathrm{len}(y)}(\mathrm{left}^{\min(n, \mathrm{len}(y))}(r_0(b), y)))).$$

There are two cases to consider: when $n \leq \mathrm{len}(y)$, which has been covered above, and when $n > \mathrm{len}(y)$. For the latter we have $\mathrm{left}^n(r_0(b), y) = \mathrm{left}^{\mathrm{len}(y)}(r_0(b), y)$, and, similarly, considering $\mathrm{right}^n(\mathrm{left}^{\mathrm{len}(y)}(r_0(b), y))$, as the right argument of right_g^n will have zero length and its left argument length that of y, we can repeat the argument for this shifting. □

The point of the last result is that, when the head is at the leftmost end of the tape, we need not be sensitive to the number of iterations. This insensitivity allows an implementation using a **do...while** construct with reversal and termination determined by detecting the ends of the tape. Of course to detect the end of a tape one must step over the end!

Intuitively, provided the left portion of the tape is never used, what is written on it will not have any impact on the action associated with g-shifts. This is a crucial observation.

Lemma 5.10. *For* $n \leq \mathrm{len}(x)$,

$$p_1(\mathrm{right}_g^n(\mathrm{left}_g^n((y, x), c))) = p_1(\mathrm{right}_g^n(\mathrm{left}_g^n(y', x, c))).$$

Proof. Consider the equation

$$\mathrm{right}_g^{\min(n, \min(\mathrm{len}(x), m))}(\mathrm{left}_g^{\min(n, \min(\mathrm{len}(x), m))}(y, x, c))$$
$$= \mathrm{right}_g^{\min(n, \min(\mathrm{len}(x), m))}(\mathrm{left}_g^{\min(n, \min(\mathrm{len}(x), m))}(y', x, c)).$$

For $n = 0$ it is clear that the equality holds. Suppose the result holds for (n, m, y, y', x, c), then consider the result for $(s(n), m, y, y', x, c)$.

If $n \geq \min(\mathrm{len}(x), m)$, the result is obvious so the only difficulty is with $n < \min(\mathrm{len}(x), m)$. If $n < \min(\mathrm{len}(x), m)$ then $s(n) \leq \min(\mathrm{len}(x), m)$ so that the power of right_g is $s(n)$, and thus we have

$$\mathrm{right}_g(\mathrm{right}_g^n(\mathrm{left}_g^{\min(\mathrm{len}(x), m)}((y, x), c)))$$
$$= \mathrm{right}_g(\mathrm{left}^{\min(\mathrm{len}(x), m) \dot{-} n}(y, x), p_1(\mathrm{right}_g^n(\mathrm{left}_g^{\min(\mathrm{len}(x), m)}((y, x), c)))).$$

Now $\mathrm{shrink}^{\min(\mathrm{len}(x), m) \dot{-} s(n)}(x)$ has length at least 1 and so the action induced in the right shift above is simply using the entry $(\min(\mathrm{len}(x), m) \dot{-} s(n), x)$, which is certainly

independent of y. But this means that the equality is preserved by the increment to $s(n)$. Thus the result holds. □

Let $b : B \to C$ and $g : A \times C \to C$. Then define $a : R \to C$ by the composition

$$a(r) = p_1(\text{right}_g^{\text{len}(r)}(\text{left}_g^{\text{len}(r)}((r_0(c(r)), r), f(c(r))))).$$

This map first moves the head to the right end of the tape and then moves it back again. Let us simulate this for $[a_1, a_2, a_3] \in \text{list}(A)$. First consider the right shifting

$$\begin{aligned}([\,], [a_1, a_2, a_3], c) &\to ([a_1], [a_2, a_3], c)\\ &\to ([a_2, a_1], [a_3], c)\\ &\to ([a_3, a_2, a_1], [\,], c),\end{aligned}$$

now the left shifting:

$$\begin{aligned}([a_3, a_2, a_1], [\,], c) &\to ([a_2, a_1], [a_3], g(a_3, c))\\ &\to ([a_1], [a_2, a_3], g(a_2, g(a_3, c)))\\ &\to ([\,], [a_1, a_2, a_3], g(a_1, g(a_2, g(a_3, c)))).\end{aligned}$$

From this it should be clear that the effect of the left shifting is to reverse the right list onto the head of the left list. This allows the list elements to be used in the action in the correct order when right shifting. Furthermore, it is clear that doing this shifting operation more than $\text{len}(r)$ times will not effect the outcome so that a **do...while** implementation which waits to detect the end of the list on each traversal is possible.

Lemma 5.11. *Whenever* $n, m \geq \text{len}(x)$,

$$a(r) = p_1(\text{right}_g^n(\text{left}_g^m((c(r), r), f(c(r))))).$$

Proof. We may express this in the form

$$p_1(\text{right}_g^{\text{len}(r)+n}(\text{left}_g^{\text{len}(r)+m}((c(r), r), f(c(r))))$$

from which the result is immediate. □

There is another interesting advantage of this type of shifting implementation: it is economical in storage. This is because the shifting operation can be done using the list cells that are given and some pointer modification which, after sweeping left then right, is non-destructive.

We now have an alternate proof of Lemma 5.8:

Proof of Lemma 5.8. Only the second identity presents any difficulties:

$$a(r_1(a, x))$$

$$= p_1(\text{right}_g^{s(\text{len}(x))}(\text{left}_g^{s(\text{len}(x))}((r_0(c(x)), r_1(a, x)), f(c(x)))))$$

$$= p_1(\text{right}_g(\text{right}_g^{\text{len}(x)}(\text{left}_g^{\text{len}x}(\text{left}_g((r_0(c(x)), r_1(a, x)), f(c(x))))))))$$

$$= \text{right}_g(\text{right}_g^{\text{len}(x)}(\text{left}_g^{\text{len}x}((r_1(a, r_0(c(x))), x), f(c(x)))))$$

$$= p_1(\text{right}_g((r_1(a, r_0(c(x))), x), p_1(\text{right}_g^{\text{len}(x)}(\text{left}_g^{\text{len}(x)}((r_0(c(x)), x), f(c(x))))))))$$

$$= g(a, p_1(\text{right}_g^{\text{len}(x)}(\text{left}_g^{\text{len}(x)}((r_0(c(x)), x), f(c(x))))))) \qquad \square$$

These results may be interpreted to be concerned with the fact that (bounded) 'loop programs' are enough to give all the functions in a distributive list-arithmetic category. This is quite a natural interpretation as one of the ways of approaching primitive computability theory is through bounded loop programs. This work then simply generalizes this intuition to a richer type theory in which certain constructions are assumed.

The shifting method also gives an efficient way of reversing a list. The 'naive reverse' is to append the current first element onto the end of the reversed tail of the list and this is unnecessarily costly: $O(n^2)$ when compared to $O(n)$ for this reverse.

6. Recursive objects in a locos

We say that a category has *enough* of a construct if whenever one could exist it does exist. Saying that a distributive category has enough prerecursives is tantamount to saying it has the adjoints required to be a recursive category. However, there is a subtle difference as the construction, $\text{rec}(A, B)$, given by the adjoint gives a canonical way of forming the recursive object which otherwise might simply exist unconstructively.

A *locos* is an arithmetic distributive category which has a canonical construction for recursive objects. For such a category it is clear that there are enough recursives. Also, more significantly, they are parameterized making a locos a list-arithmetic category:

Proposition 6.1. *A locos is a list-arithmetic category.*

Proof. It suffices to show for the prerecursive $\text{rec}(A, 1)$ that $\text{rec}(A, 1) \times B$ is prerecursive on A and B. However, this is simple. Define the maps of a semirecursive by:

$$\langle !\,.\, r_0, i_B \rangle : B \to \text{rec}(A, 1) \times B,$$

$$\langle \langle p_0, p_1\,.\,p_0 \rangle\,.\, r_1, p_1\,.\,p_1 \rangle : A \times (\text{rec}(A, 1) \times B) \to \text{rec}(A, 1) \times B.$$

It is clear that these give the required isomorphism. Next, it is necessary to show that the appropriate shrink map is a contraction. However, this simply unwinds to

the shrink $\times i_B$ which is clearly a contraction. □

Prerecursives, and thus by implication recursives, are very well behaved. The main result of this section is that connected limits of prerecursives are prerecursive in a locos. We start by proving a result which has become part of categorical folklore: namely that preservation of connected limits is ensured when equalizers and pullbacks are preserved. Paré proved a version[7] of this in 1985; I independently stumbled upon it and proved it in [5].

Proposition 6.2 (Preservation of connected limits). *A functor* $F: \mathbf{X} \to \mathbf{Y}$ *between categories with equalizers and pullbacks preserves/creates all finite connected limits if and only if it preserves/creates equalizers and pullbacks.*

Proof. Clearly if a functor preserves/creates all connected limits then, as the pullback and the equalizer are examples, certainly they will be preserved/created.

For the converse it suffices to show that any connected limit can be constructed from equalizers and pullbacks. This construction is now sketched.

Notice that some special finite limits can easily be constructed. Multiple parallel maps can be jointly equalized by forming the equalizers of pairs and intersecting (by pullback) these equalizing subobjects. Similarly simultaneous pullbacks can be formed by constructing one of the pullbacks and forming the joint equalizer over all the other squares.

Now if $D: G \to X$ is an arbitrary finite diagram, then if a universal cone (which satisfies the obvious commuting conditions) can be found based on the nodes which are the domains of the maps of G, we will be done. First let us alter the diagram by replicating the nodes, connecting the new copy by the identity map to the old copy and moving all maps which originate at the node to originate at the new node. Thus $G = (G_0, G_1, g_0, g_1)$ becomes

$$G' = (G_0 + G_0, G_0 + G_1, \langle b_0; g_0 . b_0 \rangle, \langle b_1; g_1 . b_1 \rangle).$$

However, the interpretation of the objects and maps is unchanged

$$D'_0 = \langle D_0; D_0 \rangle \quad \text{and} \quad D'_1 = \langle D_0 . i; D_1 \rangle.$$

It is easy to see that this diagram has the same limit, but the new diagram is acyclic, has domains and codomains separated, and is still connected.

For each domain in G', we form the joint equalizer of all the parallel pairs of arrows leaving that node (there may be none in which case the joint equalizer is the identity). We then change the diagram again so that these joint equalizer objects are added with their maps to the original objects. All maps which do not originate at one of these joint equalizers can now be removed without changing the limit. The

[7] Robert Paré's result also covered infinite connected limits and required that equalizers and pullback fans must be preserved.

new diagram satisfies all the conditions of the previous diagram, and, in addition it has at most one arrow between any pair of objects. This means that each joint equalizer forms a cone over a subset of the original objects.

Observe that the cone built from an equalizer object is universal for the maps eminating from the original object. We shall say that these maps are covered by that cone: the object is to obtain a cone which covers all the maps.

The next stage in the construction must be repeated until a new domain is formed whose cone covers every original map. When this happens that object is the limit.

For each pair of domains which have arrows to common objects form the simultaneous pullback and add the joint equalizer and its composite arrows to the diagram. Note that every domain is in such a pair as the diagram is connected. Let the new domains be these simultaneous pullbacks and the new maps be the composites to the original objects. The new diagram has the same limit and satisfies all the properties of the previous diagram.

When a simultaneous pullback of the apex of two cones is formed it is not hard to see that the resulting cone covers the union of the maps covered by the original two cones. The connectedness now assures us that eventually we will obtain a cone which covers all the maps. □

Our aim now is to show that semirecursives are closed to the formation of pointwise connected limits. The way we shall do this is by proving a more general result which says that any model of a certain type of 'theory', in fact a *sketch*, is closed to the formation of pointwise connected limits. This means, of course, that the inclusion of models into the category sliced by the (primitive) sorts creates connected limits. We then shall assert that semirecursives can be formulated by such a sketch.

The idea of a sketch has been around for a while and has been the subject of several lines of investigation: it is described in [2, 3]. It provides an alternative to the logic based method of describing theories which, from a categorical perspective, has several advantages. Significantly the models always form categories and it seems that there is a clean correspondence between the properties one can expect of the models and the contents of the sketch. Furthermore, most interesting concepts can be sketched in an obviously finite way.

Definition 6.3. A *finite sketch* $\mathscr{S} = (G, I, LC, CLC)$ has four components:

- G—a finite directed graph which contains all the arrows and nodes to be used in the sketch,
- I—a finite set of identities (specified by diagrams),
- LC—a finite set of finite cones made out of components in G which under interpretation must become limit cones,
- CLC—a finite set of finite cocones made out of components in G which under interpretation must become colimit cones.

A *model* of a finite sketch is a G-diagram in some category such that the interpretation of the arrows and nodes makes the identities, cone, and cocone conditions

required by the sketch valid.

The category of models in **Sets** for any sketch (including an infinite sketch) forms an admissible category (Makkai and Paré [13]). An **FP**-*sketch* is a sketch in which the only cone conditions are discrete (products) and there are no cocone conditions. An **FL**-*sketch* is a sketch in which there are no cocone conditions to be satisfied but arbitrary cone conditions. The main result concerning **FL**-sketches is that their category of models in **Sets** has pointwise limits and initial objects [3]. An **FLS**-*sketch* is a sketch in which the only cocone conditions to be satisfied are discrete (coproduct) cocones. The main result for these is that their models in **Sets** have pointwise connected limits and quasi-initial objects [2].

Our concern is with models of an **FLS**-sketch in a locos as it is obvious that semirecursives can be described by such a sketch. While it is not the case that any of the results concerning initiality can be transferred without modification into a locos, the result concerning connected limits certainly transfers in its finite form:

Proposition 6.4. *The models of an* **FLS**-*theory in a distributive category are closed to the formation of pointwise connected limits.*

Proof. It suffices by Proposition 6.2 to show that **FLS**-models are closed to pullbacks and equalizers. The limit constructs in an **FLS**-theory clearly commute with pullbacks and equalizers. The only difficulty lies in showing that this is so for coproducts. However, as the coproducts are disjoint and stable, it is easy to show that a pullback of sums is a sum of pullbacks, that is $(f_A + f_B) \wedge (g_A + g_B)$ is naturally equivalent to $(f_A \wedge g_A) + (f_B \wedge g_B)$ the sum of the pointwise pullbacks.

Similarly the equalizer of a sums is the sum of the pointwise equalizers. □

Corollary 6.5. *Pointwise connected limits of semirecursives are semirecursive.*

As prerecursives are so closely connected to semirecursives it is reasonable to suppose that they too might be closed to connected limits. The only problem is the contraction: it must be preserved by connected limits.

Proposition 6.6. *Connected limits of prerecursives are prerecursive.*

Proof. Using Proposition 6.2 it suffices to show that this is the case for equalizers and pullbacks. As every semirecursive subobject of a prerecursive is prerecursive by Lemma 5.4 the equalizer of two maps of prerecursives is a prerecursive. It remains to show that pullbacks of prerecursives are prerecursive.

Consider the pullback of $f: F \to X$ and $g: G \to X$. The following holds for $F \times G$:

$$(\text{shrink}^{\text{len}(x)}(x), \text{shrink}^{\text{len}(y)}(y)) = (r_0(c(x)), r_0(c(y))).$$

It then is clear that

$$\langle \text{shrink} \times \text{shrink} \rangle^{\max(\text{len}(x),\,\text{len}(y))}(x, y) = (r_0(c(x)), r_0(c(y))).$$

However, when this is restricted to the pullback it shows that the shrink map of this semirecursive is a contraction and thus that the pullback is prerecursive. □

An important consideration concerns whether the list arithmetic is local in the sense that the slice category $\mathbf{X}/Y$ is list arithmetic. It is clear that being distributive and arithmetic is local in this sense. The only problem concerns whether the list arithmetic can be carried over. We know from Lemma 2.2 that for the parameterized objects, that is objects of the form $p_1^{A\times Y}$ and $p_1^{B\times Y}$, we will certainly have a recursive object. The object is, in fact, $c.p_1 : \text{rec}(A, B\times Y)\to Y$, or using the list notation $p_1.p_1 : \text{list}(A)\times(B\times Y)\to Y$. We now show how to construct arbitrary recursive objects from these.

Proposition 6.7. *A locos is a local list-arithmetic category.*

Proof. Given that $\mathbf{X}/Y$ is distributive and arithmetic it suffices to exhibit a construction of a prerecursive on an arbitrary object f and the final object i_Y.

In $\mathbf{X}/Y$ we have the following two maps:

$$\text{list}(\nabla) : p_1^{\mathbb{N}\times Y}\to p_1^{\text{list}(Y)\times Y},$$
$$\text{list}(\langle f\times i_Y\rangle) : p_1^{\text{list}(A)\times Y}\to p_1^{\text{list}(Y)\times Y},$$

where $\nabla : Y\to Y\times Y$ is the diagonal map. As these maps are functorial in the list functor (insofar as we know it exists in the slice), they are certainly maps of prerecursives. But now forming the pullback

$$\begin{array}{ccc} \mathbb{Q} & \xrightarrow{\;q\;} & p_1^{\text{list}(A)\times Y} \\ {\scriptstyle \text{len}}\downarrow & & \downarrow{\scriptstyle \text{list}(\langle f\times i_Y\rangle)} \\ p_1^{\mathbb{N}\times Y} & \xrightarrow{\;\text{list}(\nabla)\;} & p_1^{\text{list}(Y)\times Y} \end{array}$$

and using Proposition 6.6, $\mathbb{Q}$ is prerecursive in the slice on f and i_Y. □

Acknowledgment

The author gratefully acknowledges the summer support provided by the "Groupe Interuniversitaire en Études Catégoriques", McGill University, Montreal, Canada, in 1986, 1987, and 1988. He would also like to acknowledge the many helpful conversations with Leopoldo Román, François Lamarche, Robert Paré, Robert Seely, and Charles Wells, all of which contributed materially to this work. Lastly, he should like to thank Michael Barr whose interest in the problem of

proving that a locos is local forced the author to finally write down the details and whose comments have always been invaluable.

References

[1] M.A. Arbib and E.G. Manes, Arrows, structures, and functors: The categorical imperative (Academic Press, New York, 1975).

[2] M. Barr, Models of sketches. Cahiers Topologie Géom. Différentielle Catégorique 27 (1986) 93–107.

[3] M. Barr and C. Wells, Toposes, Triples and Theories, Grundlehren der Mathematischen Wissenschaften 278 (Springer, Berlin, 1984).

[4] A. Burroni, Récursivité graphique (première partie): catégorie des fonctions récursive primitive formelles, Cahiers Topologie Géom. Différentielle Catégorique 27 (1986) 49–79.

[5] J.R.B. Cockett, Locally recursive categories, preprint, Department of Computer Science, University of Tennessee, Knoxville, 1986.

[6] J.R.B. Cockett, On the decidability of objects in a locos, Contemporary Math. 92 (1989) 23–46.

[7] P. Freyd, Aspects of topoi, Bull. Austral. Math. Soc. 7 (1972) 1–76.

[8] J.A. Goguen, J.W. Thatcher and E.G. Wagner, An initial algebra approach to the specification, correctness, and implementation of abstract data-types, in: R. Yeh, ed., Current: Trend in Programming Methodology, Vol. IV (Prentice-Hall, Englewood Cliffs, NJ, 1978).

[9] R.L. Goodstein, Recursive number theory (North-Holland, Amsterdam, 1964).

[10] T. Hagino, A categorical programming language, Ph.D. dissertation, University of Edinburgh.

[11] J. Lambek, Cartesian closed categories and type λ-calculi, in: C. Cousineau, P.L. Curien and B. Robinet, eds., Combinators and Functional Programming Languages, Lecture Notes in Computer Science 242 (Springer, Berlin, 1986).

[12] J. Lambek and P. Scott, Introduction to Higher Order Categorical Logic, Cambridge Studies in Advanced Mathematics 7 (Cambridge University Press, London, 1986).

[13] M. Makkai and R. Paré, Accessible categories, to appear.

[14] S. Roland, Essai sur les mathématiques algorithmiques, Mémoire de Maitrise, Université du Quebec á Montréal, 1976.

[15] L. Román, Cartesian categories with natural numbers object, J. Pure Appl. Algebra 58 (1989) 267–278.

[16] L. Román, Categories with finite limits and a natural numbers object, J. Pure Appl. Algebra, to appear.

[17] C. Wells, A generalization of the concept of sketch, Preprint 88-90, Dept. of Mathematics and Statistics, Case Western Reserve University, Cleveland, OH 44106, 1988.