Continuous Functions on Final Coalgebras

Neil Ghani[1] Peter Hancock[2]

Department of Computer and Information Sciences,
University of Strathclyde,
Glasgow, UK

Dirk Pattinson[3]

Department of Computer Science,
Imperial College,
London, UK

Abstract

In a previous paper we gave a representation of, and simultaneously a way of programming with, continuous functions on streams, whether discrete-valued functions, or functions between streams. We also defined a combinator on the representations of such continuous functions that reflects composition. Streams are one of the simplest examples of non-trivial final coalgebras. Here we extend our previous results to cover the case of final coalgebras for a broader class of functors than that giving rise to streams. Among the functors we can deal with are those that arise from countable signatures of finite-place untyped operators. These have many applications. The topology we put on the final coalgebra for such a functor is that induced by taking for basic neighbourhoods the set of infinite objects which share a common 'prefix', *a la* Baire space. The datatype of prefixes is defined together with the set of 'growth points' in a prefix, simultaneously. This we call *beheading*. To program and reason about representations of continuous functions requires a language whose type system incorporates the dependent function and pair types, inductive definitions at types Set, $I \to \mathsf{Set}$ and $(\Sigma I : \mathsf{Set})\, \mathsf{Set}^I$, coinductive definitions at types Set and $I \to \mathsf{Set}$, as well as universal arrows for such definitions.

Keywords: Continuous functions, final coalgebras, containers

1 Introduction

Eating Streams: A stream of elements of a set A is a simple example of an "infinite object". An infinite object is an inhabitant of a final coalgebra, in this case the coalgebra A^ω of the functor $X \mapsto A \times X$ or $(A\times)$ [4]. The natural notion of map whose domain is an infinite object is that of a continuous function. Informally we can think of continuity as meaning that output is produced after only a finite amount of information about the input is known. This is naturally formalised using

[1] Email: `ng@cis.strath.ac.uk`

[2] Email: `hancock@spamcop.net`

[3] Email: `dirk@doc.ic.ac.uk`

[4] We make heavy use of section notation for infix operators.

topology: if the set A is given the discrete topology, then the product (or 'Baire') topology on A^ω is generated by the family $\{N(c) | c : A^*\}$ of basic opens where

$$
\begin{aligned}
&N : A^* \to \mathcal{P}(A^\omega)\\
&N(c) \triangleq \{\, \alpha \in A^\omega \mid \overline{\alpha}(\mathit{len}\, c) = c \,\} \,.
\end{aligned}
$$

where $\mathcal{P}(X) = X \to \mathsf{Set}$, $\mathit{len}\, c$ is the length of a list c and $\overline{\alpha}(n)$ is the truncation of a stream α after n elements. With respect to the Baire topology, a discrete-valued function $f : A^\omega \to B$ is continuous if and only if it is everywhere locally constant meaning that for all $\alpha \in A^\omega$ there exists a finite prefix $c \in A^*$ of α such that the image of f throughout $N(c)$ is the singleton $\{\, f(\alpha) \,\}$. As is well-known, any such function can be represented by a particular type of wellfounded tree, that is an element of the set $T_A(B) \triangleq (\mu\, X)\, B + X^A$. Such a tree can be interpreted as a program that consumes, or *eats*, finitely many values from a stream of A's, and then terminates yielding a value of type B. This representation of continuous functions as finite trees directly encodes the intuition that continuous functions require only a finite amount of input to produce their output.

Continuity of functions whose values are themselves streams means, informally, that each element of the output stream is produced after a finite amount of the input. Topologically, continuity means that the inverse image of a value-neighbourhood is a union of argument-neighbourhoods. In a previous paper [11], we defined a representation of continuous functions between streams as trees, proved completeness in that *every* continuous function can be represented by such a tree, and defined an operation on the representations of two continuous functions between streams that yields a representation of their composite. The trees we used for our representation were non-wellfounded trees pieced together from well-founded trees: formally, we took the set of trees to be $P_A(B) = (\nu\, X)\, T_A(B \times X)$. Such a tree can be interpreted as a program to read a finite amount of a stream of A's and then produces a B and another such tree. Again, the intuition that continuity means a function requires only a finite amount of information about the input to produce each successive element of the output is hard-wired into this notion of tree.

Generalisation: The subject of this paper is the extension of our previous work to the representation to functions on final coalgebras for the broad class of endofunctors on the category Set known as *containers*. This class of functors includes besides the functor $(A\times)$, polynomial functors as $X \mapsto 1 + 2 \times X + X^2$ or power-series functors $X \mapsto \sum_{n\in\omega} C_n \times X^n$ for some family of sets $C : \omega \to \mathsf{Set}$. It also includes all the strictly positive types and the normal functors of Girard [10]. The final coalgebra of such a functor can be obtained as the projective limit of the ω-sequence $1 \longleftarrow F(1) \longleftarrow F^2(1) \longleftarrow \ldots$ starting with a terminal object 1. In that case, the elements of the final coalgebra behave sufficiently like streams that one can topologise the final coalgebra much as Baire space. We find this streamification of the final coalgebras of containers to be both illuminating and necessary to tame the complexity of the ensuing mathematical constructions. *Containers* were introduced in [2], [1] and [3] and, importantly, are closed a plethora of operations, such as constants, $+$, $\times$, $\to$, $\cdot$, Σ, Π, and least and greatest fixed points. Working with container technology seems to be much cleaner than the alternative of working with power-series functors. With the latter, operators are collected by their cardinality which is, for most purposes, unnecessarily bureaucratic.

Metatheory and Implementation: To write type-checked programs which implement continuous functions on final coalgebras of the kind we are considering requires, in general, a dependent type system [5] This means that types may be indexed by data. We also make use of initial algebras and final coalgebras for certain endofunctors on a category of families of sets indexed over a given set. Finally, we make use of a principle known as 'induction-recursion', which guarantees an initial algebra for certain endofunctors on a category of families of sets in which the index set is not fixed. For the most part, our definitions and results can be established in a form of Martin-Löf's type theory, with one universe, with inductive definitions amounting to an indexed version of the W-type.

There are some exceptions: At some points, particularly those concerned with completeness, our results depend on classical logic, and strong forms of the axiom of choice. We regard these passages as partial justification for changing the definition of continuity (in certain cases) to a more intensional one, pivoting on an inductive definition (of covering). In connection with initial algebras and final coalgebras, most of the definitions can be made with only weakly initial algebras. Completeness results about these definitions often require *uniqueness* of the universal arrow for an initial or final construction. The use we make of induction recursion is extremely weak, and must be in some sense eliminable. What we require is only a principle for forming a universe (a family of sets) closed under a slightly unusual form of Σ-quantifier. It seems likely that the finite ordinals can serve as such a universe.

History: The basic idea for this analysis of continuity emerged from expositions ([7], [16], [12], [13]) of Brouwer's notion of choice sequence and associated principles of continuity and Bar induction. There are many notions akin to choice sequence (lawless sequence for example), and the connection between streams and choice-sequences is not entirely clear [6]. Yet the principle of Bar induction can clearly be used to model continuous functions on streams, and universal quantification over streams. What we have done is, perhaps, broaden the scope of Bar induction to admit other types of infinite object than streams. It is not surprising that this can be done, as roughly speaking we can analyse these infinite objects into streams of their approximating neighbourhoods. A different extension was made by Spector [14] to bar-recursion on streams of objects of higher, non-discrete type.

The structure of the paper is as follows: Section 2 contains preliminaries and notation, Section 3 discusses containers, and Section 4 discusses the final coalgebras of containers. Then Section 5 discusses continuous functions with discrete codomain while Section 6 discusses continuous functions whose codomains are final coalgebras. Finally, Section 7 contains our conclusions.

[5] In some particular cases, such as $X \mapsto A \times X^B$, only polymorphic recursion is needed, such as is available in the programming language Haskell. What is crucial here is that there are no coproducts; with coproducts the number of meals remaining can change in irregular ways as successive slices through an infinite object are consumed.

[6] One connection between streams and choice sequences may lie in the idea that (in the angel/demon terminology of the refinement calculus [6]) inputs are chosen by the demon, while the outputs are chosen by the angel. However, choice sequences in intuitionism can involve choices to obtain further terms using a continuous function: this seems to have no explicit counterpart in the refinement calculus.

2 Preliminaries

Initial algebras and final coalgebras: μ and ν: Let $(\mu\, X)\, F(X) = \mu F$ and $(\nu\, X)\, F(X) = \nu F$ denote the initial and final coalgebras for an endofunctor F on an ambient category of sets. In general we use *in* for the structure map into the carrier of an initial algebra. Thus $\mathit{in} : F(\mu F) \to \mu F$. Given an algebra $C, \gamma : FC \to C$, let $\mathit{fold}(C;\gamma)$, or simply $\mathit{fold}(\gamma)$, be the unique morphism $\delta : \mu F \to C$ such that

$$\delta \cdot \mathit{in} = \gamma \cdot F\delta : F(\mu F) \to C\ .$$

We use in^{-1} for $\mathit{fold}(F\ \mathit{in}) : \mu F \to F(\mu\, F)$, which is the inverse of *in*.

Ex1 Finite lists $A^* \triangleq (\mu\, X)\, 1 + A \times X$. We use $\bullet$ and $(;)$ as constructors associated with $\_^*$, so

$$\begin{tikzcd} 1 \arrow[r, "\bullet"] & A^* \\ A \times A^* \arrow[r, "(;)"] & A^* \end{tikzcd}$$

$$\mathit{in} = [\bullet|(;)] : 1 + A \times A^* \to A^*$$

Ex2 Wellfounded trees $T_A(B) \triangleq (\mu\, X)\, B + X^A$, branching over A and terminating in B. $T_A(B)$ is a bifunctor, covariant in B, and contravariant in A. $T_A : \mathsf{Set} \to \mathsf{Set}$ is the free monad over the functor $(\_)^A$ (alias $(A \to)$, known as the reader monad). We write F^* for the free monad over a functor F, so that $T_A = (A \to)^*$. T_A is also known as the tree monad. We use **Ret** and **Get** for the constructors associated with T_A. Thus

$$\begin{tikzcd} B \arrow[r, "\mathbf{Ret}"] & T_A(B) \\ (T_A(B))^A \arrow[r, "\mathbf{Get}"] & T_A(B) \end{tikzcd}$$

$$\mathit{in} = [\mathbf{Ret}|\mathbf{Get}] : B + (T_A B)^A \to T_A B$$

Ex3 A more complex example of an initial algebra is for an endofunctor on the category $\mathit{Fam}(\mathsf{Set}^=)$ with set-indexed families of sets as objects, and hom-sets consisting of pullback squares. It is provided by the family $\langle S^\natural : \mathsf{Set}, P^\natural : S^\natural \to \mathsf{Set}\rangle$ defined in section 4. A family of sets defined in this way is said to be defined by *induction-recursion*. Induction-recursion is described for example in [9] and [8].

Ex4 A different example, generalising T_A from the category Set to the category $\mathsf{Set}^{S^\natural}$, is provided by *Bar* defined in section 4. In this case the initial algebra category is that of sets over an index set $S^\natural$, and the endofunctor is an example of an *indexed* container.

Final coalgebras: In general we use *out* for the structure map on the carrier of a final coalgebra. Thus $\mathit{out} : \nu F \to F(\nu F)$. Given a coalgebra $C, \gamma : C \to FC$, we use $\mathit{unfold}(C;\gamma)$, or simply $\mathit{unfold}(\gamma)$ (sometimes called coiteration of γ) to denote the unique morphism $\delta : C \to \nu F$ such that

$$\mathit{out} \cdot \delta = F\delta \cdot \gamma : C \to F(\nu F)\,.$$

We use out^{-1} for $\mathit{unfold}(F\ \mathit{out}) : F(\nu\, F) \to \nu F$, which is the inverse of *out*. For readability, we sometimes suppress notation for the isomorphisms *out* and out^{-1}.

Ex1 Streams $A^\omega = (\nu\, X)\, A \times X$. We use *hd* and *tl* to access components of a stream. $out = \langle hd, tl\rangle : A^\omega \to A \times A^\omega$, while $out^{-1}\langle a, \alpha\rangle = a; \alpha$.

Ex2 Stream processors $P_A(B) \triangleq (\nu\, X)\, T_A(B \times X)$ is a bifunctor, covariant in B, and contravariant in A.

Ex3 A more complex example, is provided by the function *Live* in section 6. In this case the category is that of sets over an index set $S^\natural$.

Terminal sequences and topology: To construct a weakly final coalgebra νF for an endofunctor F on a category with a terminal object 1, one commonly takes the limit of a certain inverse ω-chain

$$\begin{tikzcd} 1 & F1 \arrow[l, "!"'] & F^2 1 \arrow[l, "F!"'] & \cdots \arrow[l, "F^2!"'] \end{tikzcd}$$

The limit exists in a category such as Set: take the object part of the limit to be the set of $\alpha \in \prod_{n\in\omega} F^n(1)$ that satisfy the equations $\alpha_n = F^n(!)(\alpha_{n+1})$, and take the cone of projections to be $\{\, \alpha \mapsto \alpha_m \mid m : \omega \,\}$.

For well behaved functors the limit of this chain will be a final coalgebra for F. In such a case, there is a natural topology on νF. This is in fact the topology induced by the rational-valued ultrametric distance or apartness $|\alpha - \beta| = 1/2^n$ where n is least such that $\alpha_n \neq \beta_n$. Define $\alpha \sim_0 \beta$ to be vacuously true, and $\alpha \sim_{n+1} \beta$ to mean that α and β lie within $1/2^n$ of each other. Such a sequence of shrinking equivalence relations is a *separating family* in the sense of [15] if their intersection coincides with equality.

3 Containers: $(S \lhd P)$

What are Containers? Containers are representations of certain endo-functors on a category. A (unary) container is given by a set S whose elements are conventionally called 'shapes', and a family of sets $P : S \to \mathsf{Set}$, giving the possible 'positions' $p : P(s)$ within each shape $s : S$ at which data may be accessed or stored. As an alternative to the talk of shapes and positions, one may also think of $s \in S$ as symbols for (untyped) multi-place operators, with each of which is associated a set of argument places. The data S, P represents an endofunctor $(S \lhd P)$ as follows.

$$(\lhd) : (\Pi\, S : \mathsf{Set})\, (S \to \mathsf{Set}) \to (\mathsf{Set} \to \mathsf{Set})$$
$$(S \lhd P)X \triangleq (\Sigma\, s : S)\, P(s) \to X$$

Some examples of containers are:

S	P	$(\Sigma\, s : S)\, X^{P\,s}$	$(S \lhd P)$
$1 = \{\, 0 \,\}$	$0 \mapsto 1$	X	identity
A	$a \mapsto 0$	A	constant A
A	$a \mapsto 1$	$A \times X$	tagging with A
1	$0 \mapsto \omega$	X^ω	infinite sequences
ω	$n \mapsto Fin(n)$	X^*	finite sequences
$2 = \{\, 0, 1 \,\}$	$0 \mapsto 0, 1 \mapsto 1$	$1 + X$	Haskell's 'Maybe'

Here $Fin(s)$ is the set of predecessors $\{0 \cdots (s-1)\} \subseteq \omega$ of a natural number

$s \in \omega$. Container morphisms represent natural transformations between container functors. To be precise, the category *Cont* $\triangleq$ *Fam*$(\mathsf{Set}^{\mathrm{op}})$ of containers and container morphisms is fully and faithfully embedded in the functor category $\mathsf{Set}^{\mathsf{Set}}$ by the mapping $S, P \mapsto (S \lhd P)$. The category *Cont* has a rich algebraic structure [3], [1]. It has 0 and $+$, 1 and $\times$, $\cdot$, Σ_f, Π_f, Δ_f, μ and ν, not to mention some linear logic connectives and connections with the Newton-Leibnitz differential calculus. This is the basis for a powerful theory of datastructures, and polymorphic functions.

Weighted containers: $(S \lhd_C P)$ Notions of power series are of two styles. In the first $\sum_a X^{B(a)}$, which is 'cardinal-free', we may have several summands a, a' with the same arity $B(a) = B(a')$. In the second $\sum_n C(n) \times X^n$, which is 'bureaucratic', or bean-counting, we collect together the arities by their cardinal number, and put all the information into the coefficient sets $C(n)$. The notion of a weighted container - which we introduce in this paper - is a compromise. The coefficients or weights allow us to track separately information in a shape which does not affect the position sets.

$$\lhd : (\Pi\, S : \mathsf{Set})\, (\mathcal{P}S)^2 \rightarrow (\mathsf{Set} \rightarrow \mathsf{Set})$$
$$(S \lhd_C P)X \triangleq (\Sigma\, s : S)\, C(s) \times (P(s) \rightarrow X)$$

Examples of weighted containers are:

S	$C(s)$	$P(s)$	$(S \lhd_C P)X$
1	A	1	$A \times X$
A	1	1	$A \times X$
1	A	B	$A \times X^B$
A	1	B	$A \times X^B$

The notion of a weighted container is clearly connected with the notion of a left Kan extension, namely of C along P.

Note that weighted containers support various maps

$(S \lhd_C P)X \rightarrow \Sigma(S, C)$	Ignore power of X
$(S \lhd_C P)X \rightarrow (S \lhd P)X$	Ignore coefficient
$(S \lhd_C P)X \cong (\Sigma(S, C) \lhd \langle s, \_ \rangle \mapsto P(s))X$	Move coefficient to shape

4 Final coalgebras of containers

We want to capture the idea of finite information about a point in the final coalgebra $\nu(S \lhd P)$. We express finite information in trees of bounded depth, where there are special leaf nodes (all at the same distance from the root) where new growth is possible. The trees grow (ie. the neighbourhoods shrink) 'salami style'. In each step, a layer of new nodes of depth one is grown at formerly leaf positions.

Finite neighbourhoods: $(S^\natural, P^\natural)$ We define an operator on families of sets $\langle S, P\rangle$ (with $S : \mathsf{Set}$ and $P : S \to \mathsf{Set}$). We write the result of this operation $(S^\natural, P^\natural)$. Beware! $S^\natural$ depends on P as well as S, and the very type of $P^\natural$ depends on S. [7] The type of our operator is:

$$\mathit{Fam}(\mathsf{Set}) \longrightarrow \mathit{Fam}(\mathsf{Set})$$

where $\mathit{Fam}(X) = (\Sigma\, S : \mathsf{Set})\, S \to X$. An element s of $S^\natural$ plays the role here that finite sequences play in the stream case, namely it is intended to represent a finite approximation to, or neighbourhood of points, in $\nu(S \lhd P)$. The elements of $P^\natural(s)$ are locations at which the approximation s can be further refined. The operator can be defined in several ways.

- One definition is by induction-recursion. In induction recursion, one defines a set inductively, while at the same time defining a (typically set-valued) function on that set, that may be mentioned in the inductive clauses. The solution is an initial algebra for an endofunctor on a category of families. The reason that this is an induction recursive definition of a family of sets, rather then a plain inductive definition, followed by a recursive definition of sets is that the second component of the operand family is used in the definition of the first component of the output family. For an explanation of the principles underlying induction recursion see the papers [9] [8] by Dybjer and Setzer and the references therein. The constructors and their associated decoding functions are as follows:

$$\begin{aligned}
&\bullet : S^\natural\\
&P^\natural(\bullet) \triangleq 1\\
&(\,;\,) : (\Pi\, s : S^\natural)\, (P^\natural(s) \to S) \to S^\natural\\
&P^\natural(s;\sigma) \triangleq (\Sigma\, p : P^\natural(s))\, P(\sigma(p))
\end{aligned}$$

  We think that the container of finite neighbourhoods is an interesting example of an inductive-recursive definition that arises ‘in nature’, albeit one that doesn’t exploit the peculiar power of that definitional scheme, eg no ‘negative’ quantifiers or connectives like Π or $\to$ are used in defining the values of $P^\natural$. Usually, trees are grown by constructing a root for a forest. The inductive-recursive definition allows trees to grow at the leaves. The recursion scheme this induces has a computational behaviour which is appropriate in connection with the bar theorem, when typically the ‘business end’ of a formal neighbourhood is the one at which the finer distinctions are made.

- One can also define $S^\natural, P^\natural$ by using a universe closed under Σ and 1. First define $F : \omega \to \mathit{Fam}(\mathsf{Set})$ by recursion into our universe. We write $F(n) = \langle S^\natural(n), P^\natural(n)\rangle$. In the base case

$$\begin{aligned}
&S^\natural(0) = 1,\\
&P^\natural(0, \_) = 1,\\
&F(0) = (1 \lhd 1) = \mathit{id}\ .
\end{aligned}$$

[7] A more ‘logical’ sotation might be $N(S, P)$ for the neighbourhoods $S^\natural$ and $L(S, P)$ for the locations $P^\natural$.

In the step case

$$\begin{aligned}
S^{\natural}(n+1) &= (\Sigma\, s : S^{\natural}(n))\, P^{\natural}(n,s) \to S,\\
P^{\natural}(n+1,\langle s,\sigma\rangle) &= (\Sigma\, p : P^{\natural}(n,s))\, P(\sigma(p)),\\
F(n+1) &= F(n)\cdot(S \lhd P)\ .
\end{aligned}$$

Second, set $S^{\natural} = (\Sigma\, n \in \omega)\, S^{\natural}(n)$ and $P^{\natural}\langle n,s\rangle = P^{\natural}(n,s)$. The endofunctor $S^{\natural} \lhd P^{\natural}$ is the coproduct (not colimit of a chain!) $(\Sigma\, n : \omega)\,(S \lhd P)^n$.

Of course, unless the position sets are restricted to be finite such a tree may contain more than finite information, and in that case the topology is unrealistic. The question arises of whether we can choose neighbourhoods (and the associated relations of refinement and covering) in some other way to yield a useful topology when the cardinal of the index set can be countable or worse. We think there is room for speculation — the question is more subtle than it appears at first sight.

Meals : $\nu(S{\lhd}P)$ We use the term 'meal' for an element of the final coalgebra of a container, and usually use Greek letters for meal variables and names. The foodstuff a meal most resembles is perhaps broccoli, except that unlike normal broccoli, it may have infinitely long stems. Meals are so called because (as we will see) they are 'eaten', as it were in mouthfuls, a slice at a time, when the implementation of a continuous function is run on a meal.

$$\begin{aligned}
&M : \mathsf{Set}\\
&M \triangleq \nu(S \lhd P)
\end{aligned}$$

The final coalgebra gives us the following destructors:

$$\begin{aligned}
&hd : M \to S\\
&tl : (\Pi\, \alpha : M)\, P(hd(\alpha)) \to M
\end{aligned}$$

We have $out(m) = \langle hd(m), (\lambda\, p : P(hd(m)))\, tl(m,p)\rangle$.

The elements of $S^{\natural}$ are formal neighbourhoods for points in M. We write $m \models s$ to mean that s is a neighbourhood of m. If $m \models s$, we define $m[\_]_s : P^{\natural}(s) \to M$. This gives a notion of location 'inside' m.

(i) $(m \models \bullet) \triangleq \mathit{True}$

$m[\_]_{\bullet} \triangleq \mathit{const}(m) : 1 \to M$

(*ie* $\bullet$ is the topmost neighbourhood, containing all points in the space).

(ii) $(m \models s;\sigma) \triangleq (m \models s) \wedge hd \cdot m[\_]_s = \sigma : P^{\natural}(s) \to S$

$m[\_]_{(s;\sigma)} \triangleq \langle p,p'\rangle \mapsto tl(m[p]_s, p') \ : P^{\natural}(s;\sigma) \to M$

(*ie* for any $p : P^{\natural}(s)$, the symbol/shape in m at location p is $\sigma(p)$).

We can define a reflexive and transitive relation ($\sqsubseteq$) of inclusion between neighbourhoods, without quantification over M, so that $s \sqsubseteq s' \iff (\Pi\, \alpha : M)\, \alpha \models s \to \alpha \models s'$. This relation holds when s is a prefix of s', or s' is a refinement of s.

5 Representation of continuous functions with discrete co-domain

We begin by defining an inductive family of sets $Bar(B) : S^\natural \to \mathsf{Set}$, parametrised by a given family of sets $B : S^\natural \to \mathsf{Set}$. If one regards the set valued functions as predicates, the operation Bar can be recognised as a familiar closure operation that, as it were, fills in any concavity present in the predicate B (the convex closure). The very paradigm of this operation is present in Brouwer's principle of bar-induction, which explains our choice of its name. The operation can also be recognised as a form of the construction of a free monad over a functor, in this case a functor intrinsic to the topology of a final coalgebra.

$$\begin{aligned} &Bar : \mathcal{P}S^\natural \to \mathcal{P}S^\natural \\ &Bar(B) \ \triangleq\ (\mu\, X : \mathcal{P}S^\natural)\,(\lambda\, s : S^\natural) \\ &\qquad\qquad\qquad B(s) + (\Pi\, \sigma : P^\natural(s) \to S)\, X(s;\sigma) \end{aligned}$$

If $B : \mathcal{P}S^\natural$ and $s : S^\natural$, then an element of $Bar(B, s)$ is a program that implements (via the function *eat* below) a continuous function from the product space $M^{P^\natural(s)}$ to the discrete space $(\Sigma\, s \in S^\natural)\, B(s)$. The point is that we have to define the entire family of representation functions simultaneously. Crucially, it is the use of the dependent function space that makes this possible.

Just as $T_A\ \_$ is the free monad over the functor $\_^A$, Bar is the free monad Bar_1^* over the following endofunctor Bar_1 on the category of families of sets indexed over $S^\natural$.

$$\begin{aligned} &Bar_1 : \mathcal{P}S^\natural \to \mathcal{P}S^\natural \\ &Bar_1(B, s) \ \triangleq\ (\Pi\, \sigma : P^\natural(s) \to S)\, B(s;\sigma)\ . \end{aligned}$$

If we regard $S^\natural$ as a category with edges between s and $s;\sigma$ for $\sigma : P^\natural(s) \to S$, then we can shed more light (categorically) on $Bar_1 B$. It is simply $1 \to B$ where the exponential is of objects in the category $S^\natural \to \mathsf{Set}$ and thus Bar is simply $(1 \to)^*$ since Bar is just

$$(\mu\, X : \mathcal{P}S^\natural)\, B + Bar_1(X)\ .$$

where the coproduct is again in $S^\natural \to \mathsf{Set}$ and is thus taken pointwise. As a predicate, $Bar_1(B)$ says of s that B holds in one refinement step, while $Bar(B, s)$ [8] says of s that it is *barred* by B, meaning that B will inevitably hold sooner or later in any sufficiently prolonged sequence of successive refinements. One says that any m with $m \models s$ inevitably 'enters the bar' [9], *ie* there is a refinement $s' \sqsubseteq s$ such that $B(s')$ and $m \models s'$.

[8] In Sambin's notation for formal topology one would write this set in the iconic form $s \lhd B$. Unfortunately, with container notation we already use the symbol $\lhd$ infix at type $(S : \mathsf{Set}) \to \mathcal{P}S \to \mathsf{Set} \to \mathsf{Set}$ for the container extension operation.

[9] No doubt to partake of an alcoholic refreshment. A monotone bar is one that you cannot leave.

Now fix a family of discrete sets $B : \mathcal{P}S^\natural$.

$$\begin{aligned}
&\mathit{eat} : (S^\natural \lhd_{\mathit{Bar}(B)} P^\natural)(M) \to (S^\natural \lhd_B P^\natural)(M)\\
&\mathit{eat}\langle s, \mathbf{Ret}(b), m\rangle = \langle s, b, m\rangle\\
&\mathit{eat}\langle s, \mathbf{Get}(\phi), m\rangle = \mathit{eat}\langle s;\sigma, \phi(\sigma), m'\rangle\\
&\qquad\qquad\qquad\qquad\quad \mathbf{where}\, \sigma : P^\natural(s) \to S\\
&\qquad\qquad\qquad\qquad\qquad\quad \sigma(p) \triangleq \mathit{hd}(m(p))\\
&\\
&\qquad\qquad\qquad\qquad\qquad\quad m' : P^\natural(s;\sigma) \to M\\
&\qquad\qquad\qquad\qquad\qquad\quad m'\langle p, p'\rangle \triangleq \mathit{tl}(m(p), p')
\end{aligned}$$

The definition of *eat* can be justified in various ways. One is by the principle of structural recursion corresponding to the inductive definition of *Bar*. In fact this is *bar-recursion* [14]. Another exploits the freeness of the monad $\mathit{Bar} = \mathit{Bar}_1^*$ on $\mathcal{P}S^\natural$. To use the latter, we define a natural transformation from the functor Bar_1 to the functor

$$\begin{aligned}
&R : \mathcal{P}S^\natural \to \mathcal{P}S^\natural\\
&R(B, s) \ \triangleq\ M^{P^\natural(s)} \to (S^\natural \lhd_B P^\natural)(M)
\end{aligned}$$

The latter is the underlying functor of a monad. This monad can be seen as a generalisation in some sense of the simply-typed state monad $B \mapsto Q \to (B \times Q)$ where the state Q is $\{\, M^{P^\natural s} \mid s : S^\natural \,\}$.

It is convenient to introduce a notion of 'untidy' eating, that simply throws away the residue, or lets it fall to the floor. When a machine has run into its final state, we are interested only in the result.

$$\begin{aligned}
&\mathit{eat}^\dagger : (\Pi\, s : S^\natural)\, \mathit{Bar}(B, s) \to M^{P^\natural(s)} \to (\Sigma\, s : S^\natural)\, B(s)\\
&\mathit{eat}^\dagger = (\langle s, b, \_\rangle \mapsto \langle s, b\rangle) \cdot \mathit{eat}
\end{aligned}$$

Definition 5.1 If $s : S^\natural$, and $\phi : M^{P^\natural(s)} \to (\Sigma\, s : S^\natural)\, B(s)$ is extensionally equal to $\mathit{eat}^\dagger(s, t)$ for some $t : \mathit{Bar}(B, s)$, then t is said to *represent* ϕ at s, or to be a representative of it. The *representation* is the function $\mathit{eat}^\dagger$.

Theorem 5.2 (Completeness of representation for discrete codomain) *Given $s : S^\natural$, and a function $\phi : M^{P^\natural(s)} \to (\Sigma\, s : S^\natural)\, B(s)$ with no representative $t : \mathit{Bar}(B, s)$ at s, we can 'construct' (ie. prove the classical existence of) $\alpha : M$ such that $\alpha \models s$ and ϕ is not continuous at $\alpha[\_]_s$.*

6 Representatives of continuous functions with general codomain.

We consider now the problem of finding data-structures that can represent (using a suitable interpreter) functions between final coalgebras of containers. However the task is more complicated than it at first appears. Let $\langle S, P\rangle$ and $\langle Q, R\rangle$ be a pair of containers. In order to represent functions of type

$$\nu(S \lhd P) \longrightarrow \nu(Q \lhd R)$$

we need to devise a representation for an entire doubly indexed family of such functions, namely

$$(P^{\natural}(s) \to \nu(S \lhd P)) \longrightarrow (R^{\natural}(q) \to \nu(Q \lhd R))$$

where $s : S^{\natural}$ and $q : Q^{\natural}$. The reason is that we want our representation to be *compositional.* As our representation 'munches' its way through an infinite object in $\nu(S \lhd P)$, it is faced at successive stages with an indexed family of subobjects, having a type of the form

$$P^{\natural}(s) \to \nu(S \lhd P)$$

where $s : S^{\natural}$ is the prefix of the infinite object that has been eaten so far. Consider now the *output* side of our program, and imagine now that there is *another* such program, 'upstream', expecting output of successive 'slices' through an infinite object in $\nu(Q \lhd R)$. Our representation has to produce successive slices of this kind.

The interpreter eat_{∞} we need to define will have type

$$(\Pi\, q : Q^{\natural}, s : S^{\natural})\, Pos\, q\, s \to (P^{\natural}\, s \to \nu(S \lhd P)) \to (R^{\natural}\, q \to \nu(Q \lhd R))$$

where Pos is a doubly indexed family of sets, that we have yet to define, whose type is

$$Pos : Q^{\natural} \to S^{\natural} \to \mathsf{Set}\ .$$

Here is an isomorphic variant of the type our interpreter will inhabit

$$[(\Sigma\, q : Q^{\natural})\, (S^{\natural} \lhd_{Pos\, q} P^{\natural})(\nu(S \lhd P)) \times R^{\natural}\, q] \to \nu(Q \lhd R)$$

Since this type is a function space with a codomain which is a final coalgebra $\nu(Q \lhd R)$, we will define it using the universal property of final coalgebras. So we set

$$C : \mathsf{Set} \triangleq (\Sigma\, q : Q^{\natural})\, (S^{\natural} \lhd_{Pos\, q} P^{\natural})(\nu(S \lhd P)) \times R^{\natural}\, q$$

and define

$$eat_{\infty} : C \to \nu(Q \lhd R)$$
$$eat_{\infty} \triangleq unfold(\gamma)$$

where the coalgebra $\gamma : C \to (Q \lhd R)C$ is yet to be defined.

It is time to define our family of programs.

Definition 6.1 Define $Pos : Q^{\natural} \to S^{\natural} \to \mathsf{Set}$ as the following final coalgebra

$$Pos \triangleq (\nu\, W : Q^{\natural} \to S^{\natural} \to \mathsf{Set})\, \{\, q : Q^{\natural} \mid Bar\{\, s' : S^{\natural} \mid (\Sigma\, \tau : R^{\natural}\, q \to Q)\, W(q;\tau)\, s' \,\}\,\}$$

This definition has been devised to yield the fixed point isomorphism (for $q : Q^{\natural}$, at type $S^{\natural} \to \mathsf{Set}$):

$$Pos\, q \cong \quad Bar\{\, s' : S^{\natural} \mid (\Sigma\, \tau : R^{\natural}\, q \to Q)\, Pos\, (q;\tau)\, s' \,\}$$

The direction of this isomorphism we shall need is the function

$$out : (\Pi\, q : Q^{\natural}, s : S^{\natural})\, Pos(q,s) \to Bar(\{\, s' : S^{\natural} \mid (\Sigma\, \tau : R^{\natural}(q) \to Q)\, Pos(q;\tau, s')\,\}, s)$$

Note that on account of this isomorphism discrete eating (section 5) gives us a function inhabiting

$$(S^\natural \lhd_{Pos\, q} P^\natural)(\nu(S \lhd P)) \longrightarrow (S^\natural \lhd_{\{\, s:S^\natural \,|\, (\Sigma\, \tau : R^\natural\, q \to Q)\; Pos\, (q;\tau)\, s \,\}} P^\natural)(\nu(S \lhd P))$$

What remains now is to define our coalgebra $\gamma : C \to (Q \lhd R)C$. The idea is perhaps best conveyed first with a diagram.

$$\begin{tikzcd}
C \triangleq (\Sigma\, q : Q^\natural)\,(S^\natural \lhd_{Pos\, q} P^\natural)(\nu(S \lhd P)) \times R^\natural\, q \arrow[d, "eat"] \\
(\Sigma\, q : Q^\natural)\,(S^\natural \lhd_{\{\, s:S^\natural \,|\, (\Sigma\, \tau : R^\natural\, q \to Q)\; Pos\, (q;\tau)\, s \,\}} P^\natural)(\nu(S \lhd P)) \times R^\natural\, q \arrow[d, "{\langle q, s, \tau, p, f, r\rangle \mapsto \langle \tau\, r, (\lambda\, r')\, \langle q; \tau, s, p, f, \langle r, r'\rangle\rangle\rangle}"] \\
(\Sigma\, q : Q)\, R\, q \to (\Sigma\, q' : Q^\natural)\,(S^\natural \lhd_{Pos\, q'} P^\natural)(\nu(S \lhd P)) \times R^\natural\, q' \arrow[d, "\triangleq"] \\
(Q \lhd R)C
\end{tikzcd}$$

The definition can be written more formally as follows.

$$\begin{aligned}
&\gamma : C \to (Q \lhd R)C \\
&\gamma \langle q, \langle s, p, f\rangle, r\rangle \\
&\quad \triangleq \langle \tau(r), (\lambda\, r' : R(\tau(r)))\, \langle q; \tau, eat\langle s, out(q, s, p), f\rangle, \langle r, r'\rangle\rangle
\end{aligned}$$

Thus we have the coalgebra we want and hence the representation of continuous functions between final coalgebras.

We have not proved that this representation is complete, but conjecture that this can be accomplished along the lines of the proof of the simpler completeness result in section 3.3 of our earlier paper on stream processors.

6.1 Composition

In the previous section, we defined a doubly indexed family of datatypes

$$\{\, Pos\, q\, s \mid q : Q^\natural, s : S^\natural \,\}$$

to represent functions in the doubly indexed family of function types

$$\{\, (P^\natural\, s \to \nu(S \lhd P)) \to (R^\natural\, q \to \nu(Q \lhd R)) \mid q : Q^\natural, s : S^\natural \,\}$$

together with a function eat_∞ that interprets a datastructure as a function of the appropriate type. Our representations should consume successively deeper 'slices' of type $P^\natural\, s \to S$ through an infinite object of type $\nu(S \lhd P)$ presented to it as input, and produce successively deeper such slices of type $R^\natural\, q \to Q$ through an infinite object of type $\nu(Q \lhd R)$, generated by it as output. The form of the representation was motivated by the goal that our representation should be compositional: it should be possible to arrange two such representations of appropriate type in series so as to represent the composition of the represented functions.

In this section, we define an operation ('$\otimes$') directly on our datastructures that represents the composition of the functions they represent. Suppose we have three container functors $(Q \lhd R)$, $(S \lhd P)$ and $(U \lhd T)$, giving the types $\nu(Q \lhd R)$ of our ultimate output object, $\nu(S \lhd P)$ of an intermediate object, and $\nu(U \lhd T)$ of our original input object respectively. The type of the operation is as follows.

$$\otimes : (\Pi\, q : Q^\natural, s : S^\natural, u : U^\natural)\, Pos\, q\, s \to Pos\, s\, u \to Pos\, q\, u$$

To alleviate notation, we leave the first three arguments of $\otimes$ implicit, and write it as an infix operator between its final two arguments. We aim that

$$eat_\infty(post \otimes_{q,s,u} pre) = eat_\infty post \cdot eat_\infty pre$$

where $q : Q^\natural$, $s : S^\natural$, $u : U^\natural$, $post : Pos\, q\, s$ and $pre : Pos\, s\, u$. Although the underlying idea is quite simple, the details of the definition are necessarily rather technical, so we merely sketch how the function is defined. Since the codomain of the operation $\otimes$ is a final coalgebra of a certain functor on doubly indexed families, we define the operation using an *unfold*. The functor at issue is that which takes a doubly indexed family $U : Q^\natural \to U^\natural \to \mathsf{Set}$ to

$$\{\, (\Sigma\, f : R^\natural q \to Q)\, U(q; f)\, s \mid q : Q^\natural, u : U^\natural \,\}$$

, and so we seek a coalgebra

$$\gamma : (\Pi\, q : Q^\natural, u : U^\natural)\, C\, q\, u \to (\Sigma\, f : R^\natural q \to Q)\, C(q; f)\, s$$

for this functor. Experience with the corresponding operator on stream functions suggests that we take the define the carrier C of our coalgebra as follows.

$$\begin{aligned} &C : Q^\natural \to U^\natural \to \mathsf{Set} \\ &C\, q\, u \;\triangleq\; (\Sigma\, s : S^\natural)\;\; Bar\{\, s' : S^\natural \mid (\Sigma\, g : R^\natural\, q \to Q)\, Pos(q; g)s' \,\}\, s \times \\ &\qquad\qquad\qquad\qquad\quad Bar\{\, u' : U^\natural \mid (\Sigma\, h : P^\natural\, s \to S)\, Pos(s; h)u' \,\}\, u \end{aligned}$$

When the coalgebra γ has been defined, we can then define $\otimes$ as follows

$$post \otimes pre \triangleq (unfold \gamma)\langle out\, post, out\, pre\rangle\ .$$

It should be emphasised that we do *not* suggest that this is the only way to define our composition operator. Indeed, there are other methods, some of which are quite elegant, though not entirely straightforward to describe.

It remains to define the coalgebra γ. Since the domain of this function has as its principal feature two occurrences of the family transformer *Bar*, we define it using *fold*, which is to say bar-recursion. The form of the definition is with an outer recursion on the postponent, and an inner recursion on the preponent. Precise definitions are given in the accompanying type code - here we describe in operational terms what this amounts to. We evaluate the postponent to weak head normal form, which will be either **Ret**, or **Get**. In the former case (the base of the outer recursion), the arguments of the constructor have type $(\Sigma\, g : R^\natural\, q \to Q)\, Pos(q; g)s'$ for some $s' : S^\natural$. In that case, we form an element of γ's codomain by emitting the 'slice' $g : R^\natural\, q \to Q$, and forming the new state by applying *out* to the body of the Σ (so as to get an element of the postponent's type, and pairing this value with the preponent. In the latter case, when the postponent has form **Get**, we turn our attention to the preponent, and evaluate it to weak head normal form. Should the preponent have the form **Ret**, with an argument of type $(\Sigma\, h : P^\natural\, s \to S)\, Pos(s; h)u'$ for some $u' : U^\natural$, we feed the 'slice' h to the argument of the postponents **Get** (this 'going down' in the outermost recursion), thus making an internal communication from preponent to postponent, and proceed with what remains. Should the preponent on the other hand have the form **Get**, then both postponent and preponent are reading and the value of γ also has **Get** form.

It should be noted that the operator we have defined is *lazy*, in that if the postponent is ready to output a 'slice' of $\nu(Q \lhd R)$, so is the component, whether or not the preponent is ready to input a slice of $\nu(S \lhd P)$. Another variant can be defined which is *greedy*, in that as long as the preponent is ready to read, the composite will read, whether or not the postponent is prepared to write. The definition is however still a nested recursion of the same general form: an outer recursion on the postponent, with inner recursion on the preponent.

In fact, the definitions at which we arrived are merely an elaboration of the corresponding definitions in section 4.1 of our previous paper on stream processors [11]. A formal definition, type checked in a version of Agda can be found at the following url: `http://www.cis.strath.ac.uk/~ng/cont-eat.agda`. We have not actually carried out a formal proof of the equation above expressing the correctness of the operator $\otimes$, but we expect that this can be done along the lines of section 4.2 of our paper on stream processors, albeit with heavier technical machinery.

7 Conclusion

We have defined a system of representatives for functions on final coalgebras of functors expressible as containers. When those functors are finitary, their final coalgebras support a simple topology. The notion of continuity then makes sense for functions on such arguments. We proved that our representatives implement exactly the continuous functions in the discrete valued case, and conjecture it in the case that the values are in final coalgebras for finitary container. We have also defined a combinator on representatives that represents composition of the represented functions. This involved a careful analysis of the types involved. We used types indexed over neighbourhoods for the final coalgebra of the container. The mathematical techniques involve working with indexed families of sets, to represent functors F via containers, and to represent neighbourhoods of νF using in particular a weak form of inductive-recursion to define such families. We also used mixed inductive and coinductive types in our work.

As with streams, our representations are not unique. There are many different representations of even the identity function. Indeed, representations of the identity function are among the most interesting, as they correspond to non-reordering sequential channels, buffering, or wires. One might say that a representation *denotes* the extensional identity function, but *expresses* a buffering policy. It seems probable that one can define a equivalence relation directly between representations to coincide exactly with extensional equality of functions on streams.

Final coalgebras of indexed containers impose a form of sort-constraint on infinite terms (in which the indices are type identifiers). Our definition of composition has the smell of cut-elimination, except that no sort-structure is present. With a sort-structure, the source of this aroma can be investigated. There may be connections here with continuous cut-elimination and continuous normalisation and the repetition rule of Mints: see also [5].

It is striking that we now have an analysis of continuous functions on spaces of the form $F^{\infty}B = (\nu\, X)\, B \times F(X)$, *ie* cofree comonad over F. These spaces, and other such as $(\nu\, X)\, F(B \times X)$ can be useful for modelling the states of hierarchically accessed stores such as file-systems or sequentially accessed stores such as the tape of a Turing machine. Equipped with a cursor (a 'one-hole context' [4]) that navigates up and down the tree, one has an updatable store, that can be used for shared-memory communication. A storage device is a particularly well-behaved kind of state-machine, and indeed a state machine can be identified with an infinite object of a type such as $(\nu\, X)\, S \times (P \to X) = (P \to)^{\infty} S$ (in the case of a Moore machine with output S and input P). An alternative is $(\nu\, X)\, P^S \times X^S = (S \to)^{\infty}(S \to P)$ for a Mealy machine with input S and output P. Another kind of infinite object that we can now model is $(\nu\, X)\, A + X$, which has the connotation: maybe I'll give you a A, maybe I won't. When combined with a suitable account of 'hiding' delay, this may useful for modelling partial functions that may 'fail' to produce output. Finally, it is possible that one can formulate higher order eating to analyse continuity at higher types.

References

[1] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Categories of containers. In *Proceedings of Foundations of Software Science and Computation Structures*, 2003.

[2] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Representing nested inductive types using W-types. In *Automata, Languages and Programming, 31st International Colloqium (ICALP)*, pages 59 – 71, 2004.

[3] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Containers - constructing strictly positive types. *Theoretical Computer Science*, 342:3–27, September 2005. Applied Semantics: Selected Topics.

[4] Michael Abbott, Thorsten Altenkirch, Conor McBride, and Neil Ghani. ∂ for data: Differentiating data structures. *Fundamenta Informaticae*, 65(1-2):1–28, 2004.

[5] Klaus Aehlig and Felix Joachimski. On continuous normalization. In *CSL '02: Proceedings of the 16th International Workshop and 11th Annual Conference of the EACSL on Computer Science Logic*, pages 59–73, London, UK, 2002. Springer-Verlag.

[6] Ralph-Johan Back and Joakim von Wright. *Refinement calculus, A systematic introduction.* Graduate Texts in Computer Science. Springer-Verlag, New York, 1998.

[7] M. Dummett. *Elements of intuitionism.* Clarendon Press, Oxford, 2000. 2nd edition.

[8] Peter Dybjer and Anton Setzer. Induction-recursion and initial algebras. *Annals of Pure and Applied Logic*, 124:1 – 47, 2003.

[9] Peter Dybjer and Anton Setzer. Indexed induction-recursion. *Journal of Logic and Algebraic Programming*, 66:1 – 49, 2006.

[10] J-Y. Girard. Normal functors, power series and lambda-calculus. *Ann. Pure Appl. Logic*, 37(2):129–177, 1988.

[11] Peter Hancock, Neil Ghani, and Dirk Pattinson. Representations of stream processors using nested fixed points. *Logical Methods in Computer Science*, 2009 (to appear).

[12] G. Kreisel and A.S. Troelstra. Formal systems for some branches of intuitionistic analysis. *Annals of Pure and Applied Logic*, 1:229–387, 1970. Addendum in *APAL* 3, pp. 437–439.

[13] Per Martin-Löf. *Notes on Constructive Mathematics.* Almquist and Wiksell, Stockholm, 1965.

[14] Clifford Spector. Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles in current intuitionistic mathematics. In F. D. E. Dekker, editor, *Proc. Symposia in Pure Mathematics: Recursive Function Theory*, volume 5, pages 1–27. American Mathematical Society, 1962.

[15] Viggo Stoltenberg-Hansen, Ingrid Lindström, and Edward R. Griffor. *Mathematical theory of domains.* Cambridge University Press, New York, NY, USA, 1994.

[16] A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics.* North-Holland, 1988. 2 volumes.