A Coalgebraic View of Bar Recursion and Bar Induction

Venanzio Capretta[1] and Tarmo Uustalu[2]

[1] School of Computer Science, University of Nottingham, United Kingdom
[2] Institute of Cybernetics, Tallinn University of Technology, Estonia
vxc@cs.nott.ac.uk, tarmo@cs.ioc.ee

Abstract. We reformulate the bar recursion and induction principles in terms of recursive and wellfounded coalgebras. Bar induction was originally proposed by Brouwer as an axiom to recover certain classically valid theorems in a constructive setting. It is a form of induction on non-wellfounded trees satisfying certain properties. Bar recursion, introduced later by Spector, is the corresponding function definition principle.
We give a generalization of these principles, by introducing the notion of barred coalgebra: a process with a branching behaviour given by a functor, such that all possible computations terminate.
Coalgebraic bar recursion is the statement that every barred coalgebra is recursive; a recursive coalgebra is one that allows definition of functions by a coalgebra-to-algebra morphism. It is a framework to characterize valid forms of recursion for terminating functional programs. One application of the principle is the tabulation of continuous functions: Ghani, Hancock and Pattinson defined a type of wellfounded trees that represent continuous functions on streams. Bar recursion allows us to prove that every *stably* continuous function can be tabulated to such a tree where by stability we mean that the modulus of continuity is also continuous.
Coalgebraic bar induction states that every barred coalgebra is wellfounded; a wellfounded coalgebra is one that admits proof by induction.

1 Introduction

Bar induction is a reasoning principle formulated by L. E. J. Brouwer [23, 9].

He argued that it is justified in a constructive view of mathematics. Some classical theorems that are not otherwise provable in intuitionistic mathematics, follow from bar induction.

Intuitively, it posits a link between termination and induction. It says that if processes of a certain class always terminate, then the class admits a form of wellfounded induction.

Specifically, here is the original formulation of bar induction. Assume:

- Q is a decidable predicate on lists of natural numbers and Q is a *bar* (for every stream σ there exists a natural n such that the list of the first n elements of σ satisfies Q);

– R is another predicate on lists of naturals, such that Q implies R (every list satisfying Q also satisfies R) and is *inductive* (for every list l, if every extension of l by one element satisfies R, then l also satisfies R);

Then the empty list satisfies R.

In this statement, streams must be interpreted as *choice sequences*, that is, infinite sequences of natural numbers not necessarily given by an effective rule. Kleene proved that bar induction implies that not all streams are computable [16]. Recently, Nakata, Bezem and one of us (Uustalu) [17] proved some interesting consequences of the principle on the relationships between some temporal-operators (some of them mixed inductive/coinductive) on branching processes.

Later, Spector [19] formulated the principle of *bar recursion* that allows definition of higher-order functionals (see also [10] and Chapter 5 of [4]); he exploited the principle to give a proof of consistency of mathematical analysis. Berardi, Bezem and Coquand [5] reformulated Spector's principle and results; their version was termed modified bar recursion by Berger and Oliva [6], who also gave an analysis of the relation between the different variations.

We offer a reformulation and generalization of the principle in terms of coalgebras. Coalgebras are useful to model branching computational processes and types of infinite data. In previous work [7, 8] we studied the notions of *recursive* and *wellfounded coalgebras*, useful to analyze recursive functions in total functional programming and induction proofs of their properties.

Now we formulate a new notion of *barred coalgebra*, which characterizes processes whose computations always terminate, or data structures whose paths are all finite. We then state two principles as modern versions of Brouwer's.

Coalgebraic bar recursion states that every barred coalgebra is recursive. This allows us to define total recursive functions on a structure if we know that all its paths are finite. We give one application to continuous functions on streams. Ghani, Hancock and Pattinson [13] defined a type of inductive trees that represent continuous functions on streams by tabulating the outputs in their leaves. We restrict our attention to *stably* continuous functions, where *stability* means that the modulus of continuity is also continuous with itself as the modulus. We show that bar recursion implies that all stably continuous functions on streams can be tabulated.

Coalgebraic bar induction states that every barred coalgebra is wellfounded. This allows us to reason about the process by a form of induction. The original form of bar induction is an instantiation of this version.

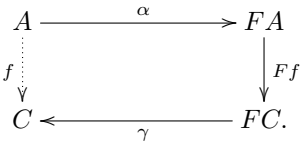
We do not claim that the principles should be accepted always and in full generality. Instead, we view them as plausible assumptions encapsulating powerful function definition and reasoning devices that need to be justified depending on the exact setting where they are invoked.

The paper is organized as follows. In Section 2, we introduce coalgebraic bar recursion. In Section 3, we apply it to tabulation of stably continuous stream functions. Coalgebraic bar induction is introduced and compared to Brouwer's principle in Section 4.

2 Bar Recursion

In previous work with Vene [7], we contributed to the study of *recursive coalgebras*, introduced by Osius [18] and further elaborated by Paul Taylor [20–22]. More recently, Adámek, Milius and colleagues [2, 3] have made additional significant contributions.

Definition 1. *A coalgebra (A, α) of an endofunctor F on a category $\mathcal{C}$ is called* recursive *if for every algebra (C, γ) there exists a unique map $f : A \to C$ (a coalgebra-to-algebra morphism) making the following diagram commute:*



This is a useful notion in total functional programming: It guarantees that every structured recursive diagram (that is, a coalgebra-to-algebra morphism diagram) based on it is a definite description of a function.

In the definition, unique existence of a mediating morphism is demanded upfront. No "more intrinsic" property of the coalgebra is invoked to guarantee unique constructibility of the solution.

We are interested in coalgebras of functors that can be viewed as tree generators with a good notion of a path. A special class of set functors, called *container functors* [1] (they are closely related to *polynomial functors* [12]), give tree types that work for us.

Definition 2. *A* container *is given by a set S of* shapes *and an S-indexed family of* positions *for every shape. It defines a set functor $[\![S, P]\!]$ by*

$$[\![S, P]\!]\, X = \Sigma s : S.\, P\, s \to X.$$

We use containers to describe "branching types" of trees. A container (S, P) says that there are S many "types" of branching nodes and that nodes of type $s : S$ have $P\, s$ many children. The sets of wellfounded and non-wellfounded trees with branching type (S, P) are described as the inductive type $\mu X.\, [\![S, P]\!]\, X$ (the initial algebra) and the coinductive type $\nu X.\, [\![S, P]\!]\, X$ (the final coalgebra). For the first of these to have any elements at all, there must be at least one shape with no positions.

We demand that inhabitedness of $P\, s$ is decidable for all $s : S$.

A coalgebra (A, α) is a process with a set of states A and a transition function $\alpha : A \to [\![S, P]\!]\, A$ that generates a shape and new states in every position. Unfolding α takes a given initial state $a : A$ to a non-wellfounded tree. This is the unique coalgebra morphism from (A, α) to the final coalgebra.

Recursiveness of the coalgebra is equivalent to this tree unfolding being actually wellfounded for every initial state $a : A$, which in other words is to say that the unique coalgebra morphism to the final coalgebra factors through the

initial algebra. (This was observed by Adámek, Lücke and Milius [2], who also called this condition the halting property.)

We suggest a classically equivalent, but constructively weaker condition. This is that, for every initial state $a : A$, every *path* in the tree unfolding of α takes one from the root to a leaf in a finite number of steps.[3]

We need first some additional definitions to formalize what we mean by paths of an element of the coalgebra.

For every state a, we define the set $\mathsf{Path}_\alpha\, a$ of paths starting from it. At each stage, the path chooses a position through which to proceed. Notation: We define types and families by rules; sets of rules written with a single rule line are inductive definitions, sets of rules written with a double rule line are coinductive definitions.

$$\frac{\alpha\, a = (s, h) \quad \neg P\, s}{\mathsf{end} : \mathsf{Path}_\alpha\, a} \qquad \frac{\alpha\, a = (s, h) \quad p : P\, s \quad \pi : \mathsf{Path}_\alpha\, (h\, p)}{p \prec \pi : \mathsf{Path}_\alpha\, a}$$

The first rule states that, if we reached a shape with no positions, then we are at a leaf of the tree and the path ends. The second rule states that we can construct a path by choosing a position in the present shape and continuing from the new state given by the transition in that position. This definition is coinductive, so the path may continue forever.

Alternatively, we could define paths as functions from $\mathbb{N}$; but this would exclude finite paths, which would have to be extended by iterating a distinguished element. The coinductive presentation is therefore more natural and avoids some coding. The status of general coinductive types in constructivism is unclear. However, paths and also tree unfoldings of coalgebras of containers can be coded with inductive and function types. So we are justified in using them.

Next we inductively define finiteness of a path. A path π is considered finite ($\pi\downarrow$) if it reaches a positionless shape after a finite number of steps. The definition mirrors that of paths, but it is inductive.

$$\frac{\alpha\, a = (s, h) \quad \neg P\, s}{\mathsf{end}\downarrow} \qquad \frac{\alpha\, a = (s, h) \quad p : P\, s \quad \pi : \mathsf{Path}_\alpha\, (h\, p) \quad \pi\downarrow}{p \prec \pi\downarrow}$$

Definition 3. *A* barred coalgebra *is a coalgebra α whose all paths are finite:*

$$\forall a : A.\, \forall \pi : \mathsf{Path}_\alpha\, a.\ \pi\downarrow .$$

Note that the simpler condition that the coalgebra has no infinite paths, although classically equivalent, is constructively weaker than finiteness of all paths. Adopting this condition instead would make coalgebraic bar induction too strong; it would not follow from Brouwer's formulation anymore. Assuming decidability of inhabitation of $P\, s$, the difference of the two conditions is exactly Markov's principle.

[3] In our constructive setting, it might in fact be more appropriate to take the name 'halting' for this condition that considers individual runs one by one rather than the whole entirety of evolutions of the process at once.

We are now ready to formulate a coalgebraic version of the principle of bar recursion. It says that finiteness of all paths of a coalgebra implies that we can define functions by recursion. Classically this is provable, but constructively it is just a plausible extra axiom.

Coalgebraic Bar Recursion: Every barred coalgebra is recursive.

$$\alpha \text{ barred} \Rightarrow \alpha \text{ recursive}$$

The converse implication does not need to be assumed: it is provable.

Proposition 1. *Every recursive coalgebra of a container functor is barred.*

$$\alpha \textit{ recursive} \Rightarrow \alpha \textit{ barred}$$

Proof. Assume that a coalgebra $\alpha : A \to [\![S, P]\!]\, A$ is recursive. Then we have the unique coalgebra-to-algebra morphism $f : A \to \mu X.\, [\![S, P]\!]\, X$. For an element $a : A$, finiteness of all paths from a is proved by structural induction on $f\, a$. □

We will not define Spector's bar recursion here. (Instead we will discuss Brouwer's bar induction in detail in Section 4.) But it corresponds to instances of coalgebraic bar recursion for $S = 1 + 1$ and $P\,(\mathsf{inl}\, *) = 0$, $P\,(\mathsf{inr}\, *) = A$, with A some fixed set. This means that $[\![S, P]\!]\, X \cong 1 + (A \to X)$. The typical case is $A = \mathbb{N}$, but Spector also considered general A. This allowed him to interpret the general axiom of countable choice.

3 Continuous Functions on Streams

We illustrate the coalgebraic bar recursion principle by applying it to continuous functions on streams.

A function on streams is continuous if it only uses a finite initial segment of its input stream to determine its result.

Ghani, Hancock and Pattinson [13] defined an inductive type of trees for representing continuous functions on streams. When applied to a stream, a function can either immediately return a result, or read the next element of the stream and continue the computation. In the tree representation, an immediate return is modelled by a leaf containing the output value, a reading operation is modelled by a node that branches according to the input value.

The statement that all continuous functions can be represented as trees in this manner is not provable constructively without additional assumptions. Ghani, Hancock and Pattinson give a proof of a negative version of this statement: If a function has no tree representation, then it cannot be continuous.

We will show that, assuming coalgebraic bar recursion, the positive statement of representability becomes provable for what we call stably continuous functions. Stability is a natural condition, requiring that the modulus of continuity is also continuous, with itself as its modulus.

Let $\mathbb{S}_A$ be the type of streams (infinite sequences) of elements of type A. The equivalence relation $=_n$, for n a natural number, identifies streams that coincide on the first n elements:

$$\sigma_1 =_n \sigma_2 \quad \text{if and only if} \quad \forall i < n.\, \sigma_1(i) = \sigma_2(i).$$

Definition 4. *A function $f : \mathbb{S}_A \to B$ from streams of elements of type A to results of type B is* continuous *if*

$$\forall \sigma : \mathbb{S}_A.\ \exists n : \mathbb{N}.\ \forall \sigma' : \mathbb{S}_A.\ \sigma' =_n \sigma \to f\,\sigma' = f\,\sigma.$$

So a function f is continuous on a stream σ if the value of f only depends on the first n elements of σ, for some n. It is continuous globally if it is continuous on every stream.

This definition corresponds to $\mathbb{S}_A$ being assigned the prodiscrete and B the discrete topology. This is what is appropriate for our purposes here. It can feel limited. For $B = \mathbb{S}_A$, for instance, not even the identity function is continuous, but then with the trees considered here it cannot be tabulated either.[4]

For other purposes, other choices can be appropriate. Brouwer's continuity principle states that all functions of type $\mathbb{S}_{\mathbb{N}} \to \mathbb{N}$ are continuous. It seems justified by a computational view of functions as programs: a terminating program computes its result in a finite number of steps; during the computation it can only read a finite number of entries from the stream. This principle can be generalized to functions $A \to B$ where A and B are assigned their "native" topologies (whereby stream types must get product topologies). However, Escardó recently discovered that the continuity principle is inconsistent in type theory [11], so the principle cannot be added safely to current type-theoretic foundations. The intuitive reason is that adding the principle adds new functions to the system and some of those are problematic. Various strands of research are investigating weaker and more refined versions of the principle that may be safe.

For our goals, it is important that a continuous function has an explicit *modulus of continuity*, the mapping that gives the length of the initial segment of the stream needed for the computation, and that this modulus is stable, that is, it makes consistent choices for streams that have the same initial segment.

Definition 5. *A* modulus of continuity *for a function $f : \mathbb{S}_A \to B$ is a function $m_f : \mathbb{S}_A \to \mathbb{N}$ such that*

$$\forall \sigma, \sigma' : \mathbb{S}_A.\ \sigma' =_{m_f\,\sigma} \sigma \to f\,\sigma' = f\,\sigma.$$

The modulus is stable *if*

$$\forall \sigma, \sigma' : \mathbb{S}_A.\ \sigma' =_{m_f\,\sigma} \sigma \to m_f\,\sigma' = m_f\,\sigma.$$

[4] In a different work [14], Hancock, Pattinson and Ghani used a mixed inductive/coinductive type to tabulate stream processors, i.e., continuous functions $f : \mathbb{S}_A \to \mathbb{S}_B$ where both $\mathbb{S}_A$ and $\mathbb{S}_B$ are given the prodiscrete topology.

So a modulus of continuity is stable if and only if it is its own modulus of continuity. Stability is a reasonable assumption in the computational view of functions:

- If, to compute $f\,\sigma$, we only need to read the first $m_f\,\sigma$ elements of σ; and
- σ' coincides with σ on the first $m_f\,\sigma$ elements;
- then we only need the first $m_f\,\sigma$ elements of σ' to compute $f\,\sigma'$ (which is equal to $f\,\sigma$);
- so it is reasonable to expect that $m_f\,\sigma' = m_f\,\sigma$.

However, given a possibly non-stable modulus, it is not in general possible to construct another modulus which is stable. In the case that the original modulus is continuous, there is an algorithm to stabilize it.

Given a non-stable but continuous modulus m, we construct a new stable modulus $\bar{m}$ for the same function. We do it by truncating every stream at an appropriate point, dictated by m, and filling in the rest of it with a fixed stream. If A is inhabited, so we know an element, we can repeat that element in a constant stream σ_0. For every σ and every index $i : \mathbb{N}$, let $\sigma|_i$ be its truncation at position i, that is, the list $[\sigma(0), \ldots, \sigma(i-1)]$. Let $n_i = m\,(\sigma|_i +\!\!+ \sigma_0)$. (The notation $+\!\!+$ denotes prepending a list to a stream and also to a path.) We now define the result of the new modulus $\bar{m}$ on σ:

$$\bar{m}\,\sigma = n_k \quad \text{where } k = \min\{i \mid n_i \leq i\}.$$

Continuity of m guarantees that $\bar{m}$ is well defined: k always exists. In fact we know that there is a j such that, if $\sigma' =_j \sigma$, then $m\,\sigma' = m\,\sigma$. In particular, if we choose $\sigma' = \sigma|_i +\!\!+ \sigma_0$ where $i = \max(j, m\,\sigma)$, we have:

$$n_i = m\,\sigma' = m\,\sigma \leq i.$$

So, since there is at least one i such that $n_i \leq i$, there is a minimal one k.

In Ghani, Hancock and Pattinson's work [13], a continuous function $f : \mathbb{S}_A \to B$ is represented by a wellfounded tree, an element of the inductive type $\mathsf{SF}_{A,B}$ given by the rules

$$\frac{b : B}{\mathsf{write}\,b : \mathsf{SF}_{A,B}} \qquad \frac{g : A \to \mathsf{SF}_{A,B}}{\mathsf{read}\,g : \mathsf{SF}_{A,B}}$$

The idea is that an element of $\mathsf{SF}_{A,B}$ is a tabulation of all the values of a function as leaves in a tree whose finite paths have to be matched against the input stream. The application of a tabulation gives us a continuous function:

$$\begin{aligned}
&\mathsf{apply} : \mathsf{SF}_{A,B} \to \mathbb{S}_A \to B\\
&\mathsf{apply}\,(\mathsf{write}\,b)\,s = b\\
&\mathsf{apply}\,(\mathsf{read}\,g)\,(a \prec s) = \mathsf{apply}\,(g\,a)\,s
\end{aligned}$$

The operator apply is defined by structural recursion on its tree argument.

Is there an inverse transformation, that is, a tabulation operator assigning a tree to every continuous function,

$$\mathsf{tabulate} : (\mathbb{S}_A \to B) \to \mathsf{SF}_{A,B}?$$

Classically, we can prove that this function exists, but the proof is not constructive. Intuitively, the algorithm to obtain the tabulation should be the following.

Let $f : \mathbb{S}_A \to B$ be continuous. Then we can define:

$$\begin{array}{ll} \mathsf{tabulate}\, f = \mathsf{write}\, b & \text{if } f \text{ "must be" constantly } b \\ \mathsf{tabulate}\, f = \mathsf{read}\, (\lambda a.\mathsf{tabulate}\, (\lambda\sigma. f\, (a \prec \sigma))) & \text{otherwise.} \end{array}$$

It is of course undecidable whether f is constant. But if the function is continuous constructively, we also have a modulus for it and can check whether the modulus is 0 on some fixed stream (we assume A to be inhabited, so we can construct one). This a sufficient condition for constancy.

However, it is not constructively provable that this algorithm generates a wellfounded tree. If the modulus is stable, we can prove that the paths of the recursive calls generated by the second equation for f are always finite. But this is not enough to conclude that the a priori non-wellfounded tabulation tree is wellfounded.

Ghani, Hancock and Pattinson proved a negative version of the statement: they did not construct a tabulation operator, but proved that, if a function cannot be tabulated, then it is not continuous.

With bar recursion, we can conclude the positive statement, using stable continuity. We provide two distinct proofs. The first proof is local: for every stably continuous function it constructs a barred coalgebra and uses it to generate the tabulation tree. The second proof is global: it constructs a single barred coalgebra on the set of all stably continuous functions and uses it to define a tabulation operation.

3.1 Individual Tabulations

First we show how to tabulate a single stably continuous function by associating an individual barred coalgebra to it. Let $f : \mathbb{S}_A \to B$ be a function with a stable modulus of continuity $m : \mathbb{S}_A \to \mathbb{N}$. We will assume that A is inhabited and has a distinguished element, which we can repeat in a stream σ_0.

We construct a coalgebra on the functor $F\, X = B + (A \to X)$ with carrier the set of lists of elements of A:

$$\begin{array}{l} \alpha_f : \mathsf{List}_A \to B + (A \to \mathsf{List}_A) \\ \alpha_f\, l = \begin{cases} \mathsf{inl}\, (f\, (l +\!\!+ \sigma_0)) & \text{if } m\, (l +\!\!+ \sigma_0) \leq \mathsf{length}\, l \\ \mathsf{inr}\, (\lambda a.\, l +\!\!+ [a]) & \text{otherwise.} \end{cases} \end{array}$$

The coalgebra checks whether the list l is sufficient to determine the result given by f: if the modulus of continuity on a stream starting with l is at most the length of l, then we know that f will depend only on it. In this case, the coalgebra terminates with the value given by f. Else, it branches into new processes for all elements of A, lengthening the list by appending the element at the end.

Observe that the functor F is a container with shapes $B + 1$, a shape for each possible leaf in B, and a single shape for the continuation. The shapes in B have no positions: the coalgebra terminates. The single continuation shape has

positions for all possible input elements, so the set of positions is A. The paths in $\mathsf{Path}_\alpha\, l$ are sequences of position choices, so in this case they are sequences of elements of A. Any such path π can be padded out to a stream of elements of A by appending σ_0 to it: $\mathsf{pad}\,\mathsf{end} = \sigma_0$, $\mathsf{pad}\,(a \prec \pi) = a \prec \mathsf{pad}\,\pi$.

Lemma 1. *α_f is a barred coalgebra.*

Proof. Given $l : \mathsf{List}_A$ and $\pi : \mathsf{Path}_\alpha\, l$, we prove that $\pi \downarrow$ by induction on $m\,\sigma_{l,\pi} - \mathsf{length}\, l$ where $\sigma_{l,\pi} = l +\!\!+ \mathsf{pad}\,\pi$.

- If $m\,\sigma_{l,\pi} - \mathsf{length}\, l = 0$, then $m\,\sigma_{l,\pi} \leq \mathsf{length}\, l$, so $\sigma_{l,\pi} =_{m\,\sigma_{l,\pi}} l +\!\!+ \sigma_0$. By stability, $m\,(l +\!\!+ \sigma_0) = m\,\sigma_{l,\pi} \leq \mathsf{length}\, l$. Therefore $\alpha_f\, l = \mathsf{inl}\,(f\,(l +\!\!+ \sigma_0))$ and $\pi = \mathsf{end}$. In this case obviously the path is finite.
- Otherwise, if $m\,\sigma_{l,\pi} - \mathsf{length}\, l > 0$, then $m\,\sigma_{l,\pi} > \mathsf{length}\, l$. It is impossible that $m\,(l +\!\!+ \sigma_0) \leq \mathsf{length}\, l$ because, if it were, then also $m\,\sigma_{l,\pi} = m\,(l +\!\!+ \sigma_0) \leq \mathsf{length}\, l$ by stability. So $\alpha_f\, l = \mathsf{inr}\,(\lambda a.\, l +\!\!+ [a])$ and $\pi = a \prec \pi'$ for some $a : A$ and $\pi' : \mathsf{Path}_\alpha\,(l +\!\!+ [a])$.
  Note that $\sigma_{l +\!\!+ [a],\pi'} = \sigma_{l,\pi}$, so $m\,\sigma_{l +\!\!+ [a],\pi'} - \mathsf{length}\,(l +\!\!+ [a]) = m\,\sigma_\pi - \mathsf{length}\, l - 1$. Therefore $\pi' \downarrow$ by the induction hypothesis and so $\pi \downarrow$. □

We can now invoke bar induction on this coalgebra to tabulate f.

Theorem 1. *The coalgebraic bar induction principle implies that there exists an element $\mathsf{tabulate}_f : \mathsf{SF}_{A,B}$ such that $\mathsf{apply}\,(\mathsf{tabulate}_f)\,\sigma = f\,\sigma$ for every $\sigma : \mathbb{S}_A$.*

Proof. If we apply the coalgebraic bar recursion principle to the statement of Lemma 1, we obtain that α_f is a recursive coalgebra.

Notice that $\mathsf{SF}_{A,B}$ is the carrier of the initial algebra of the functor F:

$$\mathsf{SF}_{A,B} = \mu X.\, B + (A \to X).$$

The copair of the two constructors $[\mathsf{write}, \mathsf{read}]$ is the actual algebra.

We can apply the defining property of recursive coalgebras to deduce that there exists a unique function tab making the following diagram commute.

$$\begin{array}{ccc} \mathsf{List}_A & \xrightarrow{\quad\alpha_f\quad} & B + (A \to \mathsf{List}_A) \\ {\scriptstyle \mathsf{tab}}\,\vdots & & \downarrow{\scriptstyle F\,\mathsf{tab}} \\ \mathsf{SF}_{A,B} & \xleftarrow[[\mathsf{write},\mathsf{read}]]{} & B + (A \to \mathsf{SF}_{A,B}). \end{array}$$

We can now define the Ghani, Hancock and Pattinson representation of f by

$$\begin{array}{l} \mathsf{tabulate}_f : \mathsf{SF}_{A,B} \\ \mathsf{tabulate}_f = \mathsf{tab}\,[]. \end{array}$$

Its correctness can be checked simply by observing that, in general,

$$\mathsf{apply}\,(\mathsf{tab}\, l)\,\sigma = f\,(l +\!\!+ \sigma).$$

□

3.2 Global Tabulation

The second way to use coalgebraic bar recursion for tabulation is to define a coalgebra on the set of all stably continuous functions:

$$\mathcal{F} = \{(f, m) \mid m : \mathbb{S}_A \to \mathbb{N} \text{ is a stable modulus of } f : \mathbb{S}_A \to B\}.$$

We use the same functor as for the individual coalgebras of the previous section: $F\,X = B + (A \to X)$. The algebra simply tests whether a function must be constant: if it has to be, it returns its value, otherwise it branches. It does so by checking that the modulus is 0 for σ_0; it must be then be 0 for all streams by stability.

$$\begin{aligned} &\alpha : \mathcal{F} \to B + (A \to \mathcal{F}) \\ &\alpha\,(f, m) = \begin{cases} \mathsf{inl}\,(f\,\sigma_0) & \text{if } m\,\sigma_0 = 0 \\ \mathsf{inr}\,(\lambda a.\,(f_a, m_a) & \text{otherwise} \end{cases} \end{aligned}$$

where

$$f_a\,\sigma = f\,(a \prec \sigma) \quad \text{and} \quad m_a\,\sigma = m\,(a \prec \sigma) - 1.$$

In the branching case, the coalgebra reads an element of the input, a, and returns the function f_a obtained by shifting f by a. The modulus m_a of f_a is one less than the modulus of f, as we do not count the prepended element a.

Lemma 2. *The coalgebra α is barred.*

Proof. Let $\pi : \mathsf{Path}_\alpha\,(f, m)$ be a path of the coalgebra. We prove that $\pi \downarrow$ by induction on $m\,\sigma_\pi$ where $\sigma_\pi = \mathsf{pad}\,\pi$.

- If $m\,\sigma_\pi = 0$, then also $m\,\sigma_0 = 0$ by stability. So $\alpha\,(f, m) = \mathsf{inl}\,(f\,\sigma_0)$ and $\pi = \mathsf{end}$.
- If $m\,\sigma_\pi > 0$, then also $m\,\sigma_0 > 0$ by stability. So $\alpha\,(f, m) = \mathsf{inr}\,(\lambda a.\,(f_a, m_a))$. It must then be that $\pi = a \prec \pi'$ for some $a : A$ and $\pi' : \mathsf{Path}_\alpha\,(f_a, m_a)$. If we then compute the modulus of the stream associated to the tail of the path, we obtain

  $$m_a\,\sigma_{\pi'} = m\,(a \prec \sigma_{\pi'}) - 1 = m\,\sigma_\pi - 1.$$

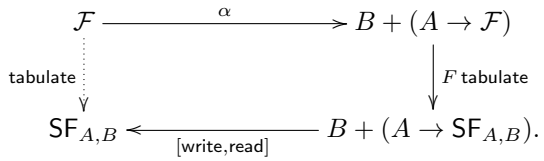
  Therefore $\pi' \downarrow$ by the induction hypothesis, so $\pi \downarrow$. □

We can now invoke bar induction on this coalgebra to construct a global tabulation operator.

Theorem 2. *The coalgebraic bar induction principle implies that there exists a function* $\mathsf{tabulate} : \mathcal{F} \to \mathsf{SF}_{A,B}$ *such that, for every continuous function with stable modulus $(f, m) : \mathcal{F}$ and every stream $\sigma : \mathbb{S}_A$, we have*

$$\mathsf{apply}\,(\mathsf{tabulate}\,(f, m))\,\sigma = f\,\sigma.$$

Proof. Applying the coalgebraic bar induction principle to the result of Lemma 2, we obtain that $(\mathcal{F}, \alpha)$ is a recursive coalgebra. In particular, the following diagram has a unique solution.



The operator $\mathsf{tabulate}$ maps every function to a correct tabulation tree for it. We prove this by induction on $m\,\sigma_0$:

– If $m\,\sigma_0 = 0$, then, by commutativity of the diagram, definition of α and continuity,

$$\begin{aligned}\mathsf{apply}\,(\mathsf{tabulate}\,(f,m))\,\sigma &= \mathsf{apply}\,([\mathsf{write},\mathsf{read}]\,(F\,\mathsf{tabulate}\,(\alpha\,(f,m))))\,\sigma\\ &= \mathsf{apply}\,([\mathsf{write},\mathsf{read}]\,(\mathsf{inl}\,(f\,\sigma_0)))\,\sigma\\ &= \mathsf{apply}\,(\mathsf{write}\,(f\,\sigma_0))\,\sigma\\ &= f\,\sigma_0 = f\,\sigma.\end{aligned}$$

– If $m\,\sigma_0 > 0$, then, writing $\sigma = a \prec \sigma'$, by commutativity of the diagram, the definition of α and the induction hypothesis,

$$\begin{aligned}\mathsf{apply}\,(\mathsf{tabulate}\,(f,m))\,\sigma &= \mathsf{apply}\,([\mathsf{write},\mathsf{read}]\,(F\,\mathsf{tabulate}\,(\alpha\,(f,m))))\,\sigma\\ &= \mathsf{apply}\,([\mathsf{write},\mathsf{read}]\,(\mathsf{inr}\,(\lambda a.\,\mathsf{tabulate}\,(f_a,m_a))))\,\sigma\\ &= \mathsf{apply}\,(\mathsf{read}\,(\lambda a.\,\mathsf{tabulate}\,(f_a,m_a)))\,(a \prec \sigma')\\ &= \mathsf{apply}\,(\mathsf{tabulate}\,(f_a,m_a)))\,\sigma'\\ &= f_a\,\sigma' = f\,(a \prec \sigma') = f\,\sigma.\end{aligned}$$

□

We finish this discussion of tabulation of continuous functions on streams by noting that we need bar recursion for tabulation because our notion of (stable) continuity of a function (which is the standard notion of continuity) is path-based. It considers one stream (choice sequence) at a time and requires that reaching an answer takes a finite number of steps. Alternatively, we could define continuity in a way that considers the entirety of possible evolutions of the choice process at once. This would result in a stronger notion of continuity and then a continuous function could be tabulated without assumptions like bar recursion.

Concretely, we could define continuity as a predicate on $\mathbb{S}_A \to B$ inductively as follows.

$$\frac{\forall \sigma, \sigma' : \mathbb{S}_A.\, f\,\sigma = f\,\sigma'}{f \text{ continuous}} \qquad \frac{\forall a : A.\, (\lambda \sigma.\, f(a \prec \sigma)) \text{ continuous}}{f \text{ continuous}}$$

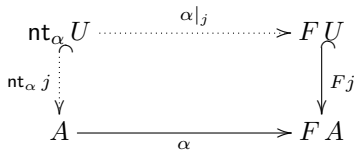
Tabulation would be immediate by structural recursion on the proof of continuity (and not exciting at all).

4 Bar Induction

We now want to give a coinductive account of the traditional formulation of bar induction. For this purpose, the notion of *wellfounded coalgebra* will be useful. Intuitively, it is a coalgebra that admits proofs of properties of its elements by induction. This notion was introduced by Taylor [20–22], who proved that, under weak reasonable assumptions, recursiveness and wellfoundedness of a coalgebra are equivalent. (In a previous article [8], we gave a review of the topic and extended it to the dual case or corecursive vs. antifounded algebras.)

Our formulation uses the *next-time operator* by Jacobs [15] on subobjects (subsets) of the carrier of a coalgebra. Let (A, α) be a coalgebra of a functor F that preserves pullbacks along monos on a category with pullbacks along monos. (These requirements are satisfied by container functors.)

Definition 6. *Let $j : U \hookrightarrow A$ be a subobject of the carrier of the coalgebra. The* next-time *subobject, $\mathsf{nt}_\alpha\, j : \mathsf{nt}_\alpha\, U \hookrightarrow A$ is defined by the following pullback.*

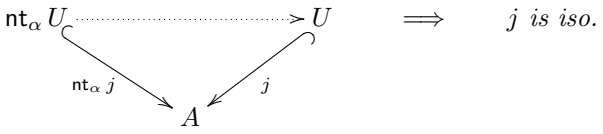


The idea of this definition is that, if U is a subset of A, then $\mathsf{nt}_\alpha\, U$ is the subset of A consisting of the elements that, after an α transition, fall into U.

In particular, suppose F is a container functor, $F = [\![S, P]\!]$. If $a : A$, then $\alpha\, a : F\, A$ has the form (s, h) with $s : S$ and $h : P\, s \to A$. We have that $a \in \mathsf{nt}_\alpha\, U$ if, for all $p : P\, s$, $h\, p \in U$.

$$\mathsf{nt}_\alpha\, U = \{a : A \mid \forall p : P\, s.\ h\, p \in U \text{ where } (s, h) = \alpha\, a\}$$

Definition 7. *The coalgebra (A, α) is* wellfounded *if, for every subobject $j : U \hookrightarrow A$, if $\mathsf{nt}_\alpha\, U$ factors through U, then j is an isomorphism. In diagram form this says that*



In simpler terms, the coalgebra is wellfounded, if $\mathsf{nt}_\alpha\, U \subseteq U$ implies $A \subseteq U$.

Intuitively, the defining property of wellfounded coalgebras is a generalization of the familiar induction principle associated to initial algebras. We want to prove that all elements of A are in the subset U. And we do this by showing that, if all "components" of an element $a : A$ (given by $\alpha\, a$) are in U, so is a.

We introduce the following principle.

Coalgebraic Bar Induction: Every barred coalgebra is wellfounded.

$$\alpha \text{ barred} \Rightarrow \alpha \text{ wellfounded}$$

This principle says that, if all paths of a coalgebra are finite, then we can prove its properties by induction.

The converse implication holds without extra assumptions.

Proposition 2. *Every wellfounded coalgebra is barred.*

$$\alpha \text{ wellfounded} \Rightarrow \alpha \text{ barred}$$

Proof. Assume that α is wellfounded and take U to be the subset of those elements of A whose all paths are finite:

$$U = \{a : A \mid \forall \pi : \mathsf{Path}_\alpha\, a.\ \pi\downarrow\}.$$

We then have that

$$\mathsf{nt}_\alpha\, U = \{a : A \mid \forall p : P\, s.\ \forall \pi' : \mathsf{Path}_\alpha\, (h\, p).\ \pi'\downarrow \text{ where } (s, h) = \alpha\, a\}.$$

Suppose $a \in \mathsf{nt}_\alpha\, U$; we want to prove that $a \in U$. So assume $\pi : \mathsf{Path}_\alpha\, a$; we will show that this path is finite. Suppose $\alpha\, a = (s, h)$. If $\pi = \mathsf{end}$, as $\neg P\, s$, then we conclude immediately that $\pi\downarrow$. If $\pi = p \prec \pi'$ for some $p : P\, s$ and $\pi' : \mathsf{Path}_\alpha\, (h\, p)$, then we know by assumption that $\pi'\downarrow$ and consequently $\pi\downarrow$.

We proved that $\mathsf{nt}_\alpha\, U \subseteq U$; therefore, by wellfoundedness of α, we have that $A \subseteq U$. So all paths are finite. □

The traditional formulation of bar induction from Brouwer is about predicates on lists of natural numbers.

Brouwer's Bar Induction:

- Let Q be a *decidable* predicate on lists of natural numbers. Assume that Q is a *bar*: $\forall \sigma : \mathbb{S}_\mathbb{N}.\ \exists n : \mathbb{N}.\ Q\, (\sigma|_n)$.
- Let R be a predicate on lists of naturals. Assume that Q implies R and R is *inductive*: $\forall l : \mathsf{List}_\mathbb{N}.\ (\forall p : \mathbb{N}.\ R\, (l +\!\!+ [p])) \to R\, l$.
- Then $R\, []$ holds.

We can assume that Q and R are suffix closed: $\forall l : \mathsf{List}_\mathbb{N}.\forall p : \mathbb{N}.\, Q\, l \to Q\, (l +\!\!+ [p])$ and similarly for R. Indeed, if they are not, we can use their suffix closures. We can define $\hat{Q}\, l$ to hold if $\exists n \leq \mathsf{length}\, l.\, Q\, (l|_n)$ and similarly for $\hat{R}$. The new predicates $\hat{Q}$ and $\hat{R}$ still satisfy the assumptions of the principle and $\hat{R}\, [] \leftrightarrow R\, []$. Also, since Q is decidable, Q being a bar implies that it is also a "tight" bar: $\forall \sigma : \mathbb{S}_\mathbb{N}.\ \exists n : \mathbb{N}.\ (\forall m : \mathbb{N}.\, m < n \to \neg Q\, (\sigma|_n)) \wedge Q\, (\sigma|_n)$.

Instead of $\mathbb{N}$, one could consider other sets in Brouwer's bar induction. The case of $\mathbb{B}$ is known as the fan "theorem" (Brouwer thought that fan theorem and bar induction were theorems, but both are just plausible axioms; the fan theorem is a positive version of weak König's lemma). In principle one could replace $\mathbb{N}$ with any fixed set, but the question is how justified this is from the constructive point of view. $\mathbb{N}$ is special in that it is a set that can be traversed by an infinite process.

The traditional form of bar induction follows from the coalgebraic version.

Theorem 3. *Coalgebraic bar induction implies Brouwer's bar induction.*

Proof. Given Q and R satisfying the assumptions of Brouwer's bar induction, we define a coalgebra structure of the functor $F\,X = 1 + (\mathbb{N} \to X)$ on the set $\mathsf{List}_{\mathbb{N}}$.

$$\alpha : \mathsf{List}_{\mathbb{N}} \to 1 + (\mathbb{N} \to \mathsf{List}_{\mathbb{N}})$$
$$\alpha\, l = \begin{cases} \mathsf{inl}\, * & \text{if } Q\, l \\ \mathsf{inr}\, (\lambda p.\, l +\!\!+ [p]) & \text{otherwise.} \end{cases}$$

The function α is welldefined, since Q is decidable. From the assumption that Q is a bar, we conclude that α is barred. Indeed, let $\pi : \mathsf{Path}_\alpha\, l$ for some list l; we must prove that π is finite. We consider the stream $\sigma = l +\!\!+ \mathsf{pad}\, \pi$ (using, e.g., 0 as the distinguished element of $\mathbb{N}$ for padding). Since Q is decidable and a bar, σ must have a shortest finite prefix $\sigma|_n$ satisfying Q. If $n < \mathsf{length}\, l$, then $\sigma|_n$ is a proper prefix of l. This forces that $\pi = \mathsf{end}$, since suffix-closedness of Q gives us that $Q\, l$. If $n \geq \mathsf{length}\, l$, then $l +\!\!+ \pi = \sigma|_n +\!\!+ \mathsf{end}$. Either way, π is finite.

Thus we can apply coalgebraic bar induction and learn that α is wellfounded.

Let $U = \{l : \mathsf{List}_{\mathbb{N}} \mid R\, l\}$. (In type theory we can use the dependent pair type $\Sigma l : \mathsf{List}_{\mathbb{N}}.\ R\, l$.) We will prove that $\mathsf{nt}_\alpha\, U \subseteq U$ from where by wellfoundedness of α it will follow that $\mathsf{List}_{\mathbb{N}} \subseteq U$.

By definition

$$\mathsf{nt}\, U = \{l : \mathsf{List}_{\mathbb{N}} \mid \alpha\, l \in 1 + (\mathbb{N} \to U)\}$$

We prove that if $l \in \mathsf{nt}\, U$, then $l \in U$, by cases:

- If $Q\, l$ holds, then $\alpha\, l = \mathsf{inl}\, *$. Since Q implies R, we also have $R\, l$, that is, $l \in U$.
- If $Q\, l$ does not hold, then $\alpha\, l = \mathsf{inr}\, (\lambda p.\, l +\!\!+ [p])$ with $l +\!\!+ [p] \in U$ for every $p : \mathbb{N}$. By the assumption that R is inductive, we can derive that $l \in U$.

Therefore, since α is wellfounded, we can deduce that $\mathsf{List}_{\mathbb{N}} \subseteq U$, that is, $\forall l : \mathsf{List}_{\mathbb{N}}.\ R\, l$.

In particular, $R\, []$, as desired. □

From Brouwer's bar induction, coalgebraic bar induction follows for the functor $F\,X = 1 + (\mathbb{N} \to X)$.

Theorem 4. *Brouwer's bar induction implies coalgebraic bar induction for* $F\,X = 1 + (\mathbb{N} \to X)$.

Proof. Given any set A with a barred coalgebra structure $\alpha : A \to 1 + (\mathbb{N} \to A)$ and a subset U of A satisfying the assumption $\mathsf{nt}_\alpha\, U \subseteq U$ of coalgebraic bar induction.

We define a function $\searrow : A \to \mathsf{List}_{\mathbb{N}} \to 1 + A$ by

$$a \searrow [] = \mathsf{inr}\, a$$
$$a \searrow (l +\!\!+ [p]) = \begin{cases} \mathsf{inl}\, * & \text{if } a \searrow l = \mathsf{inl}\, * \\ \mathsf{inl}\, * & \text{if } a \searrow l = \mathsf{inr}\, a' \text{ and } \alpha\, a' = \mathsf{inl}\, * \\ \mathsf{inr}\, (f\, p) & \text{if } a \searrow l = \mathsf{inr}\, a' \text{ and } \alpha\, a' = \mathsf{inr}\, f \end{cases}$$

For any $a : A$, we define $Q_a\, l$ to hold, if $a \searrow l = \mathsf{inl}\, *$, and $R_a\, l$ to hold, if $a \searrow l = \mathsf{inl}\, *$ or $a \searrow l = \mathsf{inr}\, a'$ and $a' \in U$.

It is immediate that, for any $a : A$, Q_a is decidable and $\forall l : \mathsf{List}_{\mathbb{N}}.\, Q_a\, l \to R_a\, l$. Moreover, as α is barred, Q_a is also a bar.

From $\mathsf{nt}_\alpha\, U \subseteq U$, it also follows that R_a is inductive for all $a : A$.

Hence, for any given $a : A$, we can apply Brouwer's bar induction, and conclude that $R_a\, []$. Since $a \searrow [] = \mathsf{inr}\, a$, this means that $a \in U$. □

Coalgebraic bar induction follows from Brouwer's bar induction also for general container functors $[\![S, P]\!]$ for which $\Sigma s : S.\, P\, s$ is isomorphic to a decidable subset of $\mathbb{N}$. This is proved by "approximate" branchings of type $[\![S, P]\!]$ with branchings of type $FX = 1 + (\mathbb{N} \to X)$.

5 Conclusions

The explicit use of bar recursion and bar induction makes it possible to encapsulate non-constructive aspects of arguments about non-wellfounded trees and stream functions.

In this paper, we have defined coalgebraic versions of bar recursion and bar induction. We find that this perspective has at least two benefits. First, we can speak of the ingredients involved in bar recursion and bar induction using the terminology of modern theoretical computer science, especially coalgebra. Second, we can avoid unnecessary coding when we want to deal with types of trees with multiple types of branching nodes, with different numbers of children.

The notion of barred coalgebra neatly fills a useful position in the range of properties that a coalgebra may satisfy. In reasoning about the total correctness of programs, it is usual to give a proof of termination for all computations and then prove by induction that some invariant holds. This application of wellfounded induction is classically justified from termination, but is not constructively valid. Bar induction is the missing principle that provides the link. Since coalgebraic methods are becoming standard tools in programming and reasoning about correctness, we hope that the coalgebraic formulation of bar recursion will establish itself as a useful tool in functional programming and type theory.

Directly applying the traditional forms of bar induction and bar recursion in concrete cases often requires a lot of encoding, namely making all nodes to be of the same type with the same number of children, which also makes all paths infinite, all this artificially; termination is then characterized by a predicate with the bar property. Our formulation is more liberal and requires no encoding: nodes of different branching degrees are allowed and termination is simply modelled by leaves.

Acknowledgements Capretta is grateful to the School of Computer Science that gave him a sabbatical semester. Uustalu was supported by the ERDF funded Estonian national CoE project EXCS and ICT national programme project Coinduction (the latter paid also Capretta's visit to Tallinn), the Estonian Science

Foundation grant no. 9475 and the Estonian Ministry of Education and Research institutional research grant no. IUT-3313.

References

1. Abbott, M., Altenkirch, T., Ghani, N.: Containers: Constructing strictly positive types. Theor. Comput. Sci. 342(1), 3–27 (2005).
2. Adámek, J., Lücke, D., Milius, S.: Recursive coalgebras of finitary functors. Theor. Inform. and Appl. 41(4), 447–462 (2007)
3. Adámek, J., Milius, S., Moss, L.S., Sousa, L.: Well-pointed coalgebras. Log. Methods in Comput. Sci. 9(3), article 2 (2013)
4. Barendregt, H.P., Dekkers, W., Statman, R.: Lambda Calculus with Types. Perspectives in Logic, Cambridge University Press (2013)
5. Berardi, S., Bezem, M., Coquand, T.: On the computational content of the axiom of choice. J. Symb. Log. 63(2), 600–622 (1998)
6. Berger, U., Oliva, P.: Modified bar recursion. Math. Struct. in Comput. Sci. 16(2), 163–183 (2006)
7. Capretta, V., Uustalu, T., Vene, V.: Recursive coalgebras from comonads. Inf. and Comput. 204(4), 437–468 (2006)
8. Capretta, V., Uustalu, T., Vene, V.: Corecursive algebras: A study of general structured corecursion. In: Oliveira, M.V.M., Woodcock, J. (eds.) SBMF 2009. LNCS, vol. 5902, pp. 84–100. Springer (2009)
9. Dummett, M.: Elements of Intuitionism. Oxford Science Publications, 2nd edn. (2000)
10. Escardó, M.H., Oliva, P.: Selection functions, bar recursion and backward induction. Math. Struct. in Comput. Sci. 20(2), 127–168 (2010)
11. Escardó, M.H., Xu, C.: The inconsistency of a Brouwerian continuity principle with the Curry-Howard interpretation. In: Altenkirch, T. (ed.) 13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015. Leibniz Int. Proc. in Informatics, vol. 38, pp. 153–164. Dagstuhl Publishing (2015)
12. Gambino, N., Hyland, M.: Wellfounded trees and dependent polynomial functors. In: Berardi, S., Coppo, M., Damiani, F. (eds.) TYPES 2003. LNCS, vol. 3085, pp. 210–225. Springer (2004)
13. Ghani, N., Hancock, P., Pattinson, D.: Continuous functions on final coalgebras. Electron. Notes in Theor. Comput. Sci. 164(1), 141–155 (2006)
14. Hancock, P., Pattinson, D., Ghani, N.: Representations of stream processors using nested fixed points. Log. Methods in Comput. Sci. 5(3), article 9 (2009)
15. Jacobs, B.: The temporal logic of coalgebras via Galois algebras. Math. Struct. in Comput. Sci. 12(6), 875–903 (2002)
16. Kleene, S., Vesley, R.: The foundations of intuitionistic mathematics: especially in relation to recursive functions. North-Holland (1965)
17. Nakata, K., Uustalu, T., Bezem, M.: A proof pearl with the fan theorem and bar induction: Walking through infinite trees with mixed induction and coinduction. In: Yang, H. (ed.) APLAS 2011. LNCS, vol. 7078, pp. 353–368. Springer (2011)
18. Osius, G.: Categorical set theory: a characterization of the category of sets. J. of Pure and Appl. Algebra 4(1), 79–119 (1974)
19. Spector, C.: Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles in current intuitionistic mathematics. In: Dekker, F.D.E. (ed.) Recursive Function Theory: Proc. of Symposia in Pure Mathematics. vol. 5, pp. 1–27. Amer. Math. Soc. (1962)

20. Taylor, P.: Intuitionistic sets and ordinals. J. of Symb. Logic 61(3), 705–744 (1996)
21. Taylor, P.: Towards a unied treatment of induction, I: The general recursion theorem. Manuscript (1996)
22. Taylor, P.: Practical Foundations of Mathematics. Cambridge Studies in Advanced Mathematics, Cambridge University Press (1999)
23. Troelstra, A.: Choice sequences. Clarendon Press (1977)