Corecursion and coinduction: what they are and how they relate to recursion and induction

Mike Gordon

Contents

This article is at http://www.cl.cam.ac.uk/~mjcg/plans/Coinduction.html and is meant to be read in a browser, but the web page can take **several minutes** to load because MathJax is slow. The PDF version is quicker to load, but the latex generated by Pandoc is not as beautifully formatted as it would be if it were from bespoke LaTeX. In this PDF version, blue text is a clickable link to a web page and pinkish-red text is a clickable link to another part of the article.

Preface

For a long time I've been aware of corecursion and coinduction as something mysteriously dual to recursion and induction, and related to maximal fixed points and bisimulation ... but I've never understood what they are or what the duality signified by the "co" means. My goal here is to try to understand these things through the activity of creating a simple explanation.

Early drafts of this article were rather formal and included proofs of everything – I even started to check some of the details with a proof assistant to try to increase my confidence that I'd got them right. After a while I realised that the writing was taking too long. Furthermore, the notational infrastructure needed for fully rigorous formal precision was obscuring the essence of the ideas. I also realised that the thing I was producing was in danger of ending up as an amateur and inferior version of existing expositions, so I decided to scale back on mathematical precision and largely give up on including formal proofs. A result of this is that I make assertions that I haven't checked in detail, so there are bound to be errors, technical mistakes and embarrassing omissions and misunderstandings. If you happen to be reading this and spot any of these, then please let me know!

To get started, I did some googling and found the link What the heck is coinduction[1] and a couple of excellent tutorials:

- *A Tutorial on Co-induction and Functional Programming*[2] by Andy Gordon.
- *An introduction to (co)algebra and (co)induction*[3] by Bart Jacobs and Jan Rutten;

I also spotted a new book *Introduction to Coalgebra*[4] by Bart Jacobs whilst browsing in the Cambridge CUP bookshop – subsequent googling discovered a preliminary version of this online here.[5]

It's a small world: in the distant past both Andy Gordon and Bart Jacobs worked as postdocs on grants for which I was one of the investigators. I think this was long before either of them became authorities on coinduction.

There is nothing new here. The material is adapted from several sources, particularly those mentioned above, as well as other papers and web pages that

[1] http://ask.metafilter.com/42858/What-the-heck-is-coinduction
[2] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.8.7706&rep=rep1&type=pdf
[3] http://homepages.cwi.nl/~janr/papers/files-of-papers/2011_Jacobs_Rutten_new.pdf
[4] http://www.cambridge.org/9781107177895
[5] http://www.cs.ru.nl/B.Jacobs/CLG/JacobsCoalgebraIntro.pdf

I stumbled across and found enlightening – links to some of these are included in the text. I mostly didn't read these beyond the introductory and motivational parts, so it's inevitable that I've misunderstood some things. As my sources are just those I happened to find, there might be better ones I could have read and cited. Please let me know if you have any suggestions.

Summary

The first three sub-sections of this summary are an example-based preview illustrating some of the core ideas covered in the rest of the article as they apply to lists. In particular:

- lists and their constructors versus
  colists and their destructors;
- recursion on finite lists versus
  corecursion on finite or infinite colists;
- the relation of least fixed points to lists and induction versus
  the relation of greatest fixed points to colists and bisimulation coinduction.

The summary concludes with a fourth sub-section that itemises the topics covered in the remainder of the article.

Recursion and induction are most familiar for natural numbers, but corecusion and coinduction for numbers are pretty useless and I couldn't find illustrative examples that didn't seem vacuous – this is why I focus on lists in this summary. However, the main part of this article is about concepts rather than examples and it starts with numbers, as they provide a minimal setting for explaining some of the core ideas. Lists can then be seen as a generalisation, as well a providing examples that hint at useful applications. Lists are also a nice stepping stone to a superficial account of the general frameworks of algebras and coalgebras and then $\mathbb{F}$-algebras and $\mathbb{F}$-coalgebras.

The material in the sections following the summary provides more explanation and details, starting with corecursion and coinduction for numbers. At the end there are brief and superficial discussions of how the number and list examples fit into the more general framework of algebras and coalgebras, and then how this relates to programming language datatypes and function definitions on these types.

Data, codata, constructors and destructors

Data represents finite values and is built by evaluating a finite number of applications of constructor functions. Codata often – but not necessarily – represents infinite values and is defined by specifying the values of destructor functions. Here are two examples.

- Data: The 4-element finite list $\mathsf{L}_4 = [1, 1, 1, 1]$ is built by specifying $\mathsf{L}_4 = \mathsf{cons}(1, \mathsf{cons}(1, \mathsf{cons}(1, \mathsf{cons}(1, \mathsf{nil}))))$, where cons is a list constructor and nil, the empty list, a nullary constructor.
- Codata: The infinite list $\mathsf{L}_\infty = [1, 1, 1, 1, \ldots]$ is defined by specifying $\mathsf{hd}(\mathsf{L}_\infty) = 1$ and $\mathsf{tl}(\mathsf{L}_\infty) = \mathsf{L}_\infty$, where hd and tl are destructors.

Recursion and corecursion

Recursion defines a function mapping values *from a datatype* by invoking itself on the components of the constructors used to build data values. Corecursion defines a function mapping *to a codatatype* by specifying the results of applying destructors to the results of the function. This is illustrated with examples using finite and infinite lists.

Let $\mathbb{N}$ be the set of natural numbers, $\mathbb{L}$ be the set of finite lists of numbers and $\mathbb{L}_\infty$ the set of infinite lists of numbers. Let $\overline{\mathbb{L}} = \mathbb{L} \cup \mathbb{L}_\infty$ be the set of finite or infinite lists of numbers.

The list constructors are nil and cons: nil is the empty list and $\mathsf{cons}(n, l)$ is the list constructed by adding the number n to the front of list l.

The list destructors are null, hd and tl: $\mathsf{null}(l)$ is true if and only $l = \mathsf{nil}$, $\mathsf{hd}(l)$ is the first element of l and $\mathsf{tl}(l)$ is the list resulting from removing its first element. The equation $l = \texttt{if } \mathsf{null}(l) \texttt{ then } \mathsf{nil} \texttt{ else } \mathsf{cons}(\mathsf{hd}(l), \mathsf{tl}(l))$ holds for every list l.

In what follows, the functions $\pi_1 : X_1 \times X_2 \to X_1$ and $\pi_2 : X_1 \times X_2 \to X_2$ extract the first and second elements of pairs: $\pi_1(x_1, x_2) = x_1$ and $\pi_2(x_1, x_2) = x_2$. The notation $A \Leftrightarrow B$ means A is true if and only if B is true.

Example. The function $\mathsf{Add1}$ adds 1 to each element of a list of numbers and is defined below both by recursion and corecursion. The recursively defined $\mathsf{Add1}$ maps finite lists to finite lists, so $\mathsf{Add1} : \mathbb{L} \to \mathbb{L}$. The corecursively defined $\mathsf{Add1}$ maps finite lists to finite lists and infinite lists to infinite lists, so $\mathsf{Add1} : \overline{\mathbb{L}} \to \overline{\mathbb{L}}$.

- Recursion:
  $\mathsf{Add1}(\mathsf{nil}) = \mathsf{nil}$ and $\mathsf{Add1}(\mathsf{cons}(n, l)) = \mathsf{cons}(n{+}1, \mathsf{Add1}(l))$
- Corecursion:
  $\mathsf{null}(\mathsf{Add1}(l)) = (l{=}\mathsf{nil})$ and $\mathsf{hd}(\mathsf{Add1}(l)) = \mathsf{hd}(l){+}1$ and $\mathsf{tl}(\mathsf{Add1}(l)) = \mathsf{Add1}(\mathsf{tl}(l))$

More generally, recursion defines functions from $\mathbb{L}$ to some set X, whereas corecursion defines functions from some set X to $\overline{\mathbb{L}}$.

- If $x_0 \in X$ and $\theta : \mathbb{N} \times X \to X$, then $f : \mathbb{L} \to X$ is defined by recursion by:
  $f(l) = \texttt{if}\ l = \mathsf{nil}\ \texttt{then}\ x_0\ \texttt{else}\ \theta(\mathsf{hd}(l), f(\mathsf{tl}(l)))$
  which specifies the result of applying f to values build using constructors:
  $f(\mathsf{nil}) = x_0$ and $f(\mathsf{cons}(n, l)) = \theta(n, f(l))$

- If $P \subseteq X$ and $\phi : X \to \mathbb{N} \times X$, then $g : X \to \overline{\mathbb{L}}$ is defined by corecursion by:
  $g(x) = \texttt{if}\ x \in P\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}(\pi_1(\phi(x)), g(\pi_2(\phi(x))))$
  which determines the result of applying the destructors to $g(x)$:
  $\mathsf{null}(g(x)) \Leftrightarrow x \in P$ and $\mathsf{hd}(g(x)) = \pi_1(\phi(x))$ and $\mathsf{tl}(g(x)) = g(\pi_2(\phi(x)))$

Returning to the $\mathsf{Add1}$ example, recall the equation:

$$\mathsf{Add1}(l) = \texttt{if}\ l = \mathsf{nil}\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}(\mathsf{hd}(l){+}1, \mathsf{Add1}(\mathsf{tl}(l)))$$

This can be viewed as a definition by recursion by taking $f = \mathsf{Add1}$, $X = \mathbb{L}$, $x_0 = \mathsf{nil}$ and $\theta(n, l) = \mathsf{cons}(n{+}1, l)$ in the recursion scheme for f in the first bullet point above.

It can be viewed as a definition by corecursion by taking $g = \mathsf{Add1}$, $X = \overline{\mathbb{L}}$, $P = \{\mathsf{nil}\}$ and $\phi(l) = (\mathsf{hd}(l){+}1, \mathsf{tl}(l))$ in corecursion scheme for g in the second bullet point.

The version of $\mathsf{Add1}$ defined by recursion maps $\mathbb{L}$ to $\mathbb{L}$, so only finite lists can be arguments and results, but the version defined by corecursion maps $\overline{\mathbb{L}}$ to $\overline{\mathbb{L}}$, which allows both finite and infinite lists to be arguments and results.

This sub-section concludes with three examples of corecursion that will reappear later.

Example. The function $\mathsf{CountFrom} : \mathbb{N} \to \overline{\mathbb{L}}$ maps a number to an infinite list, i.e. to a member of the $\mathbb{L}_\infty$ subset of $\overline{\mathbb{L}}$.

$$\mathsf{CountFrom}(n) \ = \ \mathsf{cons}(n, \mathsf{CountFrom}(n{+}1))$$

This corecursion equation defines $\mathsf{CountFrom}(n)$ to be the infinite list counting up from n, i.e. $\mathsf{cons}(n, \mathsf{cons}(n{+}1, \mathsf{cons}(n{+}2, \ldots)))$. If $X = \mathbb{N}$, P is the empty set and $\phi(n) = (n, n{+}1)$, then $\mathsf{CountFrom} = g$, where g is defined by corecursion by the single corecursion equation:

$$g(x) = \texttt{if}\ x \in P\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}(\pi_1(\phi(x)), g(\pi_2(\phi(x))))$$

$\mathsf{CountFrom}$ can also be specified by giving equations for the destructors:

$$\begin{array}{lcl} \mathsf{null}(\mathsf{CountFrom}(n)) &=& \text{false} \\ \mathsf{hd}(\mathsf{CountFrom}(n)) &=& n \\ \mathsf{tl}(\mathsf{CountFrom}(n)) &=& \mathsf{CountFrom}(n{+}1) \end{array}$$

Example. The function $\mathsf{CountUp} : \mathbb{N} \times \mathbb{N} \to \overline{\mathbb{L}}$ maps a pair of numbers (m, n) to a finite list when $m \leq n$ and to an infinite lists when $m > n$.

$$\mathsf{CountUp}(m, n) \ = \ \texttt{if}\ m = n\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}(m, \mathsf{CountUp}(m{+}1, n))$$

If $m \leq n$, then this corecursion equation defines $\mathsf{CountUp}(m, n)$ to be the finite list counting up from m until n (including m but not n, so $\mathsf{CountUp}(m, m) = \mathsf{nil}$ and $\mathsf{CountUp}(m, m{+}1) = \mathsf{cons}(m, \mathsf{nil})$).

If $m > n$, then the equation defines $\mathsf{CountUp}(m, n)$ to be the infinite list counting up from m.

If $X = \mathbb{N} \times \mathbb{N}$, $P = \{(m, n) \mid m = n\}$ and $\phi(m, n) = (m, (m{+}1, n))$, then $\mathsf{CountUp} = g$, where g is defined by corecursion by the single corecursion equation:

$$g(x) = \texttt{if } x \in P \texttt{ then } \mathsf{nil} \texttt{ else } \mathsf{cons}(\pi_1(\phi(x)), g(\pi_2(\phi(x))))$$

$\mathsf{CountUp}$ can also be specified by giving equations for the destructors:

$$\begin{array}{lcll}
\mathsf{null}(\mathsf{CountUp}(m, n)) & = & (m = n) & \\
\mathsf{hd}(\mathsf{CountUp}(m, n)) & = & m & \text{if } m \neq n \\
\mathsf{tl}(\mathsf{CountUp}(m, n)) & = & \mathsf{CountUp}(m{+}1, n) & \text{if } m \neq n
\end{array}$$

Example. The function $\mathsf{CountUpTo} : \mathbb{N} \times \mathbb{N} \to \overline{\mathbb{L}}$ maps a pair of numbers to a finite lists, i.e. to a member of the $\mathbb{L}$ subset of $\overline{\mathbb{L}}$.

$$\mathsf{CountUpTo}(m, n) \ = \ \texttt{if } m \geq n \texttt{ then } \mathsf{nil} \texttt{ else } \mathsf{cons}(m, \mathsf{CountUpTo}(m{+}1, n))$$

If $m \leq n$ then this corecursion equation defines $\mathsf{CountUp}(m, n)$ to be the finite list counting up from m until n (including m but not n. If $m > n$, then $\mathsf{CountUp}(m, n) = \mathsf{nil}$.

If $X = \mathbb{N} \times \mathbb{N}$, $P = \{(m, n) \mid m \geq n\}$ and $\phi(m, n) = (m, (m{+}1, n))$, then $\mathsf{CountUpTo} = g$, where g is defined by corecursion by the single corecursion equation:

$$g(x) = \texttt{if } x \in P \texttt{ then } \mathsf{nil} \texttt{ else } \mathsf{cons}(\pi_1(\phi(x)), g(\pi_2(\phi(x))))$$

$\mathsf{CountUpTo}$ can also be specified by giving equations for the destructors:

$$\begin{array}{lcll}
\mathsf{null}(\mathsf{CountUpTo}(m, n)) & = & (m \geq n) & \\
\mathsf{hd}(\mathsf{CountUpTo}(m, n)) & = & m & \text{if } m < n \\
\mathsf{tl}(\mathsf{CountUpTo}(m, n)) & = & \mathsf{CountUpTo}(m{+}1, n) & \text{if } m < n
\end{array}$$

Fixed points, induction, coinduction and bisimulation

A *fixed point* of a function θ is an x such that $\theta(x) = x$.

If $L \subseteq \overline{\mathbb{L}}$, define $\mathcal{F}(L) = \{\mathsf{nil}\} \cup \{\mathsf{cons}(n, l) \mid n \in \mathbb{N} \text{ and } l \in L\}$.

$\mathbb{L}$ and $\overline{\mathbb{L}}$ are both fixed points of $\mathcal{F}$, i.e. $\mathcal{F}(\mathbb{L}) = \mathbb{L}$ and $\mathcal{F}(\overline{\mathbb{L}}) = \overline{\mathbb{L}}$. The least fixed point is $\mathbb{L}$ and the greatest fixed point is $\overline{\mathbb{L}}$.

$\mathbb{L}$ is also the least *pre-fixed point* of $\mathcal{F}$, that is the least $L \subseteq \overline{\mathbb{L}}$ such that $\mathcal{F}(L) \subseteq L$. Thus if $\mathcal{F}(L) \subseteq L$ then by leastness $\mathbb{L} \subseteq L$. This is the principle of induction for finite lists since $\mathcal{F}(L) \subseteq L$ is equivalent to $\mathsf{nil} \in L$ (the base case of the induction) and $\forall n \in \mathbb{N}. \forall l \in L. \mathsf{cons}(n, l) \in L$ (the induction step). $\mathbb{L} \subseteq L$ is equivalent to $\forall l \in \mathbb{L}. l \in L$.

$\overline{\mathbb{L}}$ is also the greatest post-fixed point of $\mathcal{F}$, that is the greatest $L \subseteq \overline{\mathbb{L}}$ such that $L \subseteq \mathcal{F}(L)$, but there doesn't seem to be any interesting reasoning principle arising from this. The principle dual to induction is: if $L \subseteq \mathcal{F}(L)$ then by greatestness $L \subseteq \overline{\mathbb{L}}$ – but this is vacuous as $L \subseteq \overline{\mathbb{L}}$ is assumed.

However, there is a useful reasoning principle for binary relations on $\overline{\mathbb{L}}$ that is derived from greatest fixed points. This is the main coinduction principle and is based on subsets of $\overline{\mathbb{L}} \times \overline{\mathbb{L}}$ rather than subsets of just $\overline{\mathbb{L}}$.

If $R \subseteq \overline{\mathbb{L}} \times \overline{\mathbb{L}}$ is a binary relation on $\overline{\mathbb{L}}$, then define $\mathcal{B}(R) \subseteq \overline{\mathbb{L}} \times \overline{\mathbb{L}}$ by:

$\mathcal{B}(R) = \{(\mathsf{nil}, \mathsf{nil})\} \cup \{(\mathsf{cons}(n, l_1), \mathsf{cons}(n, l_2)) \mid n \in \mathbb{N} \wedge (l_1, l_2) \in R\}$

The least fixed point of $\mathcal{B}$ is the equality relation on $\mathbb{L}$, i.e. $\{(l, l) \mid l \in \mathbb{L}\}$, and the greatest fixed point is the equality relation on $\overline{\mathbb{L}}$, i.e. $\{(l, l) \mid l \in \overline{\mathbb{L}}\}$.

The relation $\{(l, l) \mid l \in \overline{\mathbb{L}}\}$ is also the greatest *post-fixed point* of $\mathcal{B}$, that is the greatest R such that $R \subseteq \mathcal{B}(R)$. Thus if there exists an R such that $R \subseteq \mathcal{B}(R)$, then by greatestness $R \subseteq \{(l, l) \mid l \in \overline{\mathbb{L}}\}$. This is a principle of coinduction.

The property $R \subseteq \mathcal{B}(R)$ means that if $(l_1, l_2) \in R$ then either $l_1 = \mathsf{nil}$ and $l_2 = \mathsf{nil}$ or else $l_1 = \mathsf{cons}(n, l_1')$ and $l_2 = \mathsf{cons}(n, l_2')$ and $(l_1', l_2') \in R$, for some $n \in \mathbb{N}$ and $l_1', l_2' \in \overline{\mathbb{L}}$. Such an R is called a *bisimulation*.

The proof principle that if R is a bisimulation then $\forall l_1\, l_2 \in \overline{\mathbb{L}}. (l_1, l_2) \in R \Rightarrow l_1 = l_2$ is called *bisumulation coinduction* here, or just *coinduction*, which is the more common name for this principle.

Recall the corecursively defined functions:

$$\begin{array}{lcl}
\mathsf{CountFrom}(n) & = & \mathsf{cons}(n, \mathsf{CountFrom}(n{+}1)) \\
\mathsf{CountUp}(m, n) & = & \mathtt{if}\ m = n\ \mathtt{then}\ \mathsf{nil}\ \mathtt{else}\ \mathsf{cons}(m, \mathsf{CountUp}(m{+}1, n)) \\
\mathsf{CountUpTo}(m, n) & = & \mathtt{if}\ m \geq n\ \mathtt{then}\ \mathsf{nil}\ \mathtt{else}\ \mathsf{cons}(m, \mathsf{CountUpTo}(m{+}1, n))
\end{array}$$

Examples of bisimulations are R_1 and R_2, where:

$$\begin{array}{lcl}
R_1 & = & \{(\mathsf{CountUp}(m, n), \mathsf{CountFrom}(m)) \quad \mid m > n\} \\
R_2 & = & \{(\mathsf{CountUp}(m, n), \mathsf{CountUpTo}(m, n)) \mid m \leq n\}
\end{array}$$

The easy arguments that R_1 and R_2 are bisimulations are given at the beginning of the section on examples of bisimulation coinduction for lists.

By bisimulation coinduction applied to R_1:
$\forall m\, n \in \mathbb{N}. m > n \Rightarrow \mathsf{CountUp}(m, n) = \mathsf{CountFrom}(m)$.

By bisimulation coinduction applied to R_2:
$\forall m\, n \in \mathbb{N}. m \leq n \Rightarrow \mathsf{CountUp}(m, n) = \mathsf{CountUpTo}(m, n)$.

By combining these two applications of bisimulation coinduction, the equation below can be deduced.

$\mathsf{CountUp}(m, n) = \mathtt{if}\ m \leq n\ \mathtt{then}\ \mathsf{CountUpTo}(m, n)\ \mathtt{else}\ \mathsf{CountFrom}(m)$

Both the terms "bisimulation" and "coinduction" originate from Robin Milner. For further historical details see Section 4.3 of Sangiorgi's paper On the Origins of Bisimulation and Coinduction[6] and Milner and Tofte's paper Co-induction in relational semantics[7].

Overview of the rest of this article

The rest of this paper consists of the following.

- A discussion of the traditional Peano axioms for numbers and their equivalence to another definition, which is based on the unique existence of functions and whose dual yields the conumbers.
- Numbers are fitted into the general framework of algebras and its dual, coalgebras, is introduced. Corecursion for conumbers is explained and some examples given.
- Lists are introduced as another example of algebras and colists as another example of coalgebras. Examples of corecursion for lists are given.
- Coinduction is explained and the way in which it is dual to induction discussed. Bisimulation relations are introduced and their central role in coinduction explained. Examples for numbers and lists are described.
- $\mathbb{F}$-algebras and $\mathbb{F}$-coalgebras are introduced as a uniform framework with numbers and lists being special cases. Algebra and coalgebra morphisms are defined. Initial and terminal algebras and coalgebras are explained as a general way of defining datatypes, with numbers and lists being examples.
- The roles of least and greatest fixed points in characterising numbers, conumbers, lists and colists are explained and so is the relation of fixed points to induction and coinduction.
- Some brief comments are made on how algebras and coalgebras relate to programming language datatypes, and how recursion and corecursion can be used to define function on data and codata.
- The article concludes with a brief discussion of what is and isn't covered and some reflections on what I've learnt by writing it.

I think that most of the key ideas about the relationships between recursion, induction, corecursion and coinduction have been covered in this summary section – the rest of the article is just more details and examples.

[6] http://www.cs.unibo.it/~sangio/DOC_public/history_bis_coind.pdf
[7] http://www.sciencedirect.com/science/article/pii/030439759190033X

Natural numbers, Peano's axioms and Peano structures

The Wikipedia article[8] says that coinduction is the "mathematical dual to structural induction", so a good starting place is ordinary mathematical induction, which is structural induction applied to the natural number structure $(\mathbb{N}, 0, \mathsf{S})$, where $\mathbb{N}$ is a set, $0 \in \mathbb{N}$ a constant and $\mathsf{S} : \mathbb{N} \to \mathbb{N}$ a one-argument function.

The five Peano axioms characterise the natural number structure $(\mathbb{N}, 0, \mathsf{S})$.

1. $0 \in \mathbb{N}$
2. $\forall n \in \mathbb{N}.\ \mathsf{S}(n) \in \mathbb{N}$
3. $\forall n \in \mathbb{N}.\ \mathsf{S}(n) \neq 0$
4. $\forall m \in \mathbb{N}.\ \forall n \in \mathbb{N}.\ \mathsf{S}(m) = \mathsf{S}(n) \ \Rightarrow\ m = n$
5. $\forall M.\ 0 \in M \land (\forall n \in M.\ \mathsf{S}(n) \in M) \ \Rightarrow\ \mathbb{N} \subseteq M$

The structure $(\mathbb{N}, 0, S)$ is an instance of a class of structures $(\mathcal{A}, z, s)$, where $\mathcal{A}$ is a set, $z \in \mathcal{A}$ and $s : \mathcal{A} \to \mathcal{A}$. These structures seem to have several names, including discrete dynamical systems[9] and Peano structures[10]. The latter name is used here.

The existence of $(\mathbb{N}, 0, \mathsf{S})$ satisfying Peano's axioms follows from the axioms of set theory[11], e.g. see this wikipedia article[12]. Peano's axioms entail the principle of recursive definition. This says that for any Peano structure $(\mathcal{A}, z, s)$ there is exactly one function $f : \mathbb{N} \to \mathcal{A}$ such that $f(0) = z$ and $\forall n \in \mathbb{N}. f(\mathsf{S}(n)) = s(f(n))$. Showing this is straightforward, but not entirely trivial (e.g. see here[13] for a discussion and here[14] for a detailed proof).

The principle of recursive definition is equivalent to Peano's axioms because $(\mathbb{N}, 0, \mathsf{S})$ satisfies the five Peano axioms if and only if for all Peano structures $(\mathcal{A}, z, s)$ there is exactly one function $f : \mathbb{N} \to \mathcal{A}$ such that: $f(0) = z \land \forall n \in \mathbb{N}.\ f(\mathsf{S}(n)) = s(f(n))$. $(\mathbb{N}, 0, \mathsf{S})$ an example of an initial algebra.

[8] https://en.wikipedia.org/wiki/Coinduction
[9] http://abel.math.harvard.edu/~mazur/preprints/when_is_one.pdf
[10] https://proofwiki.org/wiki/Definition:Peano_Structure
[11] https://en.wikipedia.org/wiki/Zermelo-Fraenkel_set_theory
[12] https://en.wikipedia.org/wiki/Set-theoretic_definition_of_natural_numbers
[13] http://devlinsangle.blogspot.co.uk/2011/11/how-multiplication-is-really-defined-in.html
[14] https://proofwiki.org/wiki/Principle_of_Recursive_Definition_for_Peano_Structure

Algebras and coalgebras

A Peano structure is an example of an algebra. The dual of an algebra is a coalgebra. The dual of the natural numbers are the conatural numbers, also called the conumbers. These will be described using coalgebras.

Algebras and coalgebras can be formulated in various ways. A simple one defines an algebra to be a structure consisting of a carrier set plus some number of distinguished elements and some number of functions, called constructors, whose range is the carrier set.

The dual of an algebra is a *coalgebra*. This also has a carrier set, but instead of constructor functions that build members of the carrier, a coalgebra has destructor functions that split members of the carrier into the components they are built from. The carrier of an algebra is the range of its constructors, but the carrier of a coalgebra is the domain of its destructors.

A destructor is dual to a constructor in that it splits the results of a construction into its components. If c is an n-ary constructor and $c(a_1, \ldots, a_n) = a$, then its dual destructor, d say, splits a into $(a_1, \ldots, a_n)$, i.e. $d(a) = (a_1, \ldots, a_n)$. In general, the dual of a non-nullary constructor is a partial function – it is only defined on those elements that are constructed by the dual constructor, these elements are the *domain* of the destructor. The domain of d is denoted by $\mathsf{Dom}(d)$.

The *domain* of a non-nullary destructor is the set of members of the carrier that are constructed using the dual constructor. The domain of a nullary operator is the set just containing it. The carrier of a coalgebra is required to be partitioned by the domains of its destructors. Thus each element of the carrier will be a member of the domain of exactly one destructor.

The relation between constructors and destructors described above is only intended to provide some intuition. I don't know whether in general algebras and coalgebras come in pairs with each constructor in an algebra dual to a destructor in the coalgebra it's paired with. Such a relationship may hold for some particular algebra-coalgebra pairs, like the Peano algebra and coalgebra described below (and maybe also for all initial $\mathbb{F}$-algebras and terminal $\mathbb{F}$-coalgebras). I'm ignorant of the full theory, but my guess is that such a relationship doesn't hold in general.

For scarily more abstract views, see this [15] and this[16], which are random examples I found with Google.

[15] https://www.tu-braunschweig.de/Medien-DB/iti/survey_full.pdf

[16] http://www.math.uni-duesseldorf.de/~wisbauer/algebra-coalgebra.pdf

Numbers and conumbers

As discussed in the previous section, an example of an algebra is a Peano algebra, which was also previously called a Peano structure. A Peano algebra $(\mathcal{A}, z, s)$ has a carrier $\mathcal{A}$, only one distinguished element $z \in \mathcal{A}$ and only one constructor function $s : \mathcal{A} \to \mathcal{A}$. The arity of a constructor is the number of arguments it takes. Distinguished elements like z are considered to be nullary constructors, i.e. to have arity 0.

The natural numbers are the Peano algebra $(\mathbb{N}, 0, \mathsf{S})$ with the property that for any Peano structure $(\mathcal{A}, z, s)$ there's exactly one function $f : \mathbb{N} \to \mathcal{A}$ such that $f(0) = z$ and $\forall n \in \mathbb{N}.\ f(\mathsf{S}(n)) = s(f(n))$. The function f can also be defined by a single equation $f(n) = \texttt{if } n{=}0 \texttt{ then } z \texttt{ else } s(f(n{-}1))$.

The dual of the unary number constructor S is the predecessor function P defined by $\mathsf{P}(n) = n{-}1$, where P is the partial function only defined on non-zero numbers, so $\mathsf{Dom}(\mathsf{P}) = \{n \mid n > 0\}$. Why P is the dual of S is explained below, when the conatural numbers are characterised.

Nullary constructors represent distinguished elements of the carrier, so it's not obvious what their corresponding destructors are, since there are no components of a corresponding constructor to return. To cope with this, destructors corresponding to nullary constructors return a 'dummy value' to represent 'no components'. This value is traditionally denoted by $*$, though $(\,)$ might be more mnemonic. This may seem like an odd way to represent the duals of nullary constructors, but it should become more motivated in the section on $\mathbb{F}$-algebras and $\mathbb{F}$-coalgebras below. In the summary section at the beginning of this article, $x \in d$ is an abbreviation for $x \in \mathsf{Dom}(d)$, where d is a nullary destructor.

The conumber dual of the distinguished number 0 is the partial function $\mathsf{is0} : \{0\} \to \{*\}$, and so necessarily $\mathsf{is0}(0) = *$ and $\mathsf{Dom}(\mathsf{is0}) = \{0\}$.

If a coalgebra has more than one destructor, then all its destructors are partial functions. Here's some useful notation for partial functions.

- Writing $\theta : X \nrightarrow Y$ means θ is a partial function from set X to set Y.
- The subset of X where θ is defined – the domain of θ – is denoted by $\mathsf{Dom}(\theta)$.
- If $\theta : X \to Y$ – i.e. θ is a total function – then $\mathsf{Dom}(\theta) = X$.
- If $\theta : X \nrightarrow Y$ or $\theta : X \to Y$ and if $U \subseteq X$ and $V \subseteq Y$, then writing $\theta : U \to V$ means that if $x \in U$ then $\theta(x) \in V$. In particular, if $\theta : X \nrightarrow Y$ then $\theta : \mathsf{Dom}(\theta) \to Y$.

The dual of a Peano algebra $(\mathcal{A}, z, s)$ is a Peano coalgebra $(\mathcal{C}, isz, p)$ where isz and p are destructors: $isz : \mathcal{C} \to \{*\}$ is a nullary destructor and $p : \mathcal{C} \nrightarrow \mathcal{C}$ is a unary destructor. The domains of isz and p partition $\mathcal{C}$, so if $x \in \mathcal{C}$ then either $x \in \mathsf{Dom}(isz)$ or $x \in \mathsf{Dom}(p)$, but not both.

The *conatural numbers* are the Peano coalgebra $(\overline{\mathbb{N}}, \mathsf{is0}, \mathsf{P})$ with the property that for any Peano coalgebra $(\mathcal{C}, isz, p)$ there is exactly one function $g : \mathcal{C} \to \overline{\mathbb{N}}$ such that for all $x \in \mathcal{C}$:

- if $x \in \mathsf{Dom}(isz)$ then $g(x) \in \mathsf{Dom}(\mathsf{is0})$ and $\mathsf{is0}(g(x)) = isz(x)$;
- if $x \in \mathsf{Dom}(p)$ then $g(x) \in \mathsf{Dom}(\mathsf{P})$ and $\mathsf{P}(g(x)) = g(p(x))$.

The coalgebra $(\overline{\mathbb{N}}, \mathsf{is0}, \mathsf{P})$ is an example of a terminal coalgebra.

It turns out that the unique existence of g determines the Peano coalgebra $(\overline{\mathbb{N}}, \mathsf{is0}, \mathsf{P})$ to have carrier set $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$, the nullary destructor $\mathsf{is0} : \{0\} \to \{*\}$ satisfying $\mathsf{is0}(0) = *$ and the unary destructor P to be the predecessor function $n \mapsto n{-}1$ extended by defining $\mathsf{P}(\infty) = \infty$, thus $\mathsf{P} : \{n \mid (n \in \mathbb{N} \wedge n > 0) \vee n = \infty\} \to \overline{\mathbb{N}}$.

It's useful to extend the addition operator[17] to conumbers $\overline{\mathbb{N}}$ by specifying that if either argument is ∞ then so is the result. For example, $\infty{+}1 = \infty$. Note that $+$ extended to $\overline{\mathbb{N}}$ is associative and commutative; these properties are used in examples below. With this extension, $m = \mathsf{P}(n) \Leftrightarrow m{+}1 = n$ for all m and n in $\overline{\mathbb{N}}$.

With this extended definition of $+$, the function g is defined by the single equation $g(x) = \texttt{if}\ isz(x) = *\ \texttt{then}\ 0\ \texttt{else}\ g(p(x)){+}1$.

Note the ways in which natural numbers are dual to conatural numbers:

- constructors $0 \in \mathbb{N}$, $\mathsf{S} : \mathbb{N} \to \mathbb{N}$ versus
  destructors $\mathsf{is0} : \overline{\mathbb{N}} \nrightarrow \{*\}$, $\mathsf{P} : \overline{\mathbb{N}} \nrightarrow \overline{\mathbb{N}}$;
- unique $f : \mathbb{N} \to \mathcal{A}$ versus
  unique $g : \mathcal{C} \to \overline{\mathbb{N}}$;
- define f on constructed values: $f(0) = z \wedge f(\mathsf{S}(n)) = s(f(n))$ versus
  define destructors on values of g: $\mathsf{is0}(g(x)) = isz(x) \wedge \mathsf{P}(g(x)) = g(p(x))$;
- recursion $f(n) = \texttt{if}\ n{=}0\ \texttt{then}\ z\ \texttt{else}\ s(f(n{-}1))$ versus corecursion[18]
  $g(x) = \texttt{if}\ isz(x) = *\ \texttt{then}\ 0\ \texttt{else}\ g(p(x)){+}1$.
- simple recursion on natural numbers *always* terminates versus
  corecursion on conumbers *sometimes* doesn't terminate (see example below).

[17] https://ncatlab.org/nlab/show/corecursion
[18] https://ncatlab.org/nlab/show/corecursion

Example of corecursion for numbers

The notation $[x_1 \mapsto v_1, \ldots, x_k \mapsto v_k]$ denotes the function θ with domain $\{x_1, \ldots, x_k\}$ defined by $\theta(x_i) = v_i$. Using this notation, consider the Peano coalgebra $(\mathcal{C}, isz, p)$ where:

$$\begin{array}{lcl} \mathcal{C} & = & \{A, B, C, D, E, F, G, H, I, J\}, \\ isz & = & [F \mapsto *, J \mapsto *], \\ p & = & [A \mapsto A, B \mapsto C, C \mapsto B, D \mapsto E, E \mapsto F, G \mapsto H, H \mapsto I, I \mapsto J]. \end{array}$$

$\mathsf{Dom}(isz) = \{F, J\}$ and $\mathsf{Dom}(p) = \{A, B, C, D, E, G, H, I\}$, so $\mathsf{Dom}(isz)$ and $\mathsf{Dom}(p)$ partition $\mathcal{C}$.

This coalgebra is diagrammed in Figure 1 below: an arrow from x to x' means that if $x \in \mathsf{Dom}(p)$ then $p(x) = x'$, and if $x \in \mathsf{Dom}(isz)$ then $isz(x) = *$.

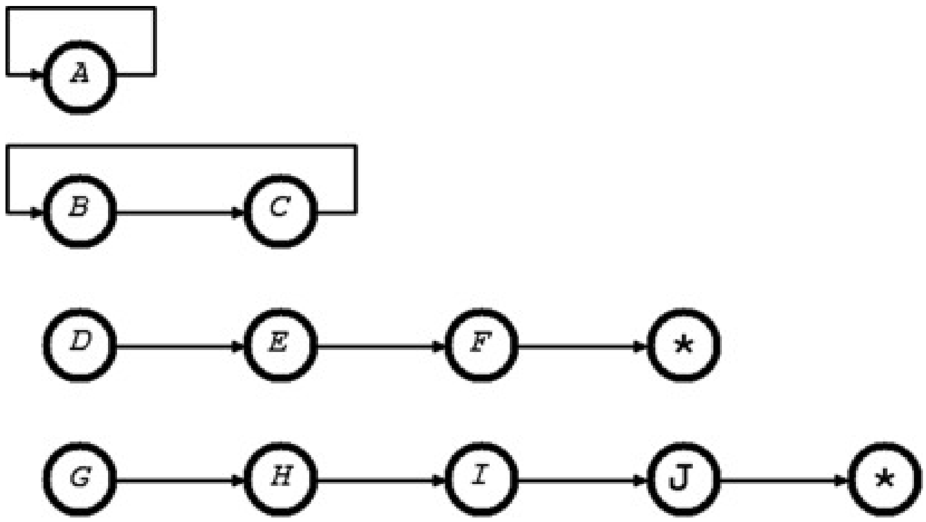

Figure 1:

Consider now the definition of $g : \mathcal{C} \to \overline{\mathbb{N}}$ specified for $x \in \mathcal{C}$ by $g(x) =$ `if` $isz(x) = *$ `then` 0 `else` $g(p(x))+1$.

Rewriting $g(A)$ with this equation yields:

$$g(A) = g(p(A))+1 = g(A)+1$$

The only way this can be satisfied is with $g(A) = \infty$.

Rewriting $g(B)$ and $g(C)$ yields:

$$g(B) = g(p(B))+1 = g(C)+1 = (g(p(C))+1)+1 = (g(B)+1)+1 = g(B)+2$$
$$g(C) = g(p(C))+1 = g(B)+1 = (g(p(B))+1)+1 = (g(C)+1)+1 = g(C)+2$$

The only way these can be satisfied is with $g(B) = \infty$ and $g(C) = \infty$.

Thus if $x \in \{A, B, C\}$ then evaluating $g(x)$ by rewriting with the equation for g loops.

On other arguments the rewriting terminates:

$g(F) = 0$
$g(E) = g(p(E))+1 = g(F)+1 = 0+1 = 1$
$g(D) = g(p(D))+1 = g(E)+1 = 1+1 = 2$

$g(J) = 0$
$g(I) = g(p(I))+1 = g(J)+1 = 0+1 = 1$
$g(H) = g(p(H))+1 = g(I)+1 = 1+1 = 2$
$g(G) = g(p(G))+1 = g(H)+1 = 2+1 = 3$

These rewriting calculations show that:

$g = [A \mapsto \infty, B \mapsto \infty, C \mapsto \infty, D \mapsto 2, E \mapsto 1, F \mapsto 0, G \mapsto 3, H \mapsto 2, I \mapsto 1, J \mapsto 0]$

For an arbitrary Peano coalgebra $(\mathcal{C}, isz, p)$, the function $g : \mathcal{C} \to \overline{\mathbb{N}}$ satisfying the equation $g(x) = \texttt{if}\ isz(x) = *\ \texttt{then}\ 0\ \texttt{else}\ g(p(x))+1$ is described by:

$$g(x) = \begin{cases} n & \exists x_0 \cdots x_n.\ x = x_0\ \wedge\ isz(x_n) = *\ \wedge\ \forall i{<}n.\, p(x_i) = x_{i+1} \\ \infty & \text{otherwise} \end{cases}$$

The example just given illustrates this.

Lists and colists

If $\mathfrak{A}$ is a set – "$\mathfrak{A}$" for "alphabet" – then the set $\mathbb{L}_{\mathfrak{A}}$ of finite lists (or strings) of members of $\mathfrak{A}$ is defined by two constructors: the empty list $\mathsf{nil} \in \mathbb{L}_{\mathfrak{A}}$ and the function $\mathsf{cons} : \mathfrak{A} \times \mathbb{L}_{\mathfrak{A}} \to \mathbb{L}_{\mathfrak{A}}$ which constructs the new list $\mathsf{cons}(\mathfrak{a}, l)$ that results from adding the element $\mathfrak{a} \in \mathfrak{A}$ to the front of list $l \in \mathbb{L}_{\mathfrak{A}}$.

Peano-style axioms for lists are:

1. $\mathsf{nil} \in \mathbb{L}_{\mathfrak{A}}$
2. $\forall \mathfrak{a} \in \mathfrak{A}.\ \forall l \in \mathbb{L}_{\mathfrak{A}}.\ \mathsf{cons}(\mathfrak{a}, l) \in \mathbb{L}_{\mathfrak{A}}$
3. $\forall \mathfrak{a} \in \mathfrak{A}.\ \forall l \in \mathbb{L}_{\mathfrak{A}}.\ \mathsf{cons}(\mathfrak{a}, l) \neq \mathsf{nil}$
4. $\forall \mathfrak{a}_1\, \mathfrak{a}_2 \in \mathfrak{A}.\ \forall l_1\, l_2 \in \mathbb{L}_{\mathfrak{A}}.\ \mathsf{cons}(\mathfrak{a}_1, l_1) = \mathsf{cons}(\mathfrak{a}_2, l_2)\ \Rightarrow\ \mathfrak{a}_1 = \mathfrak{a}_2 \wedge l_1 = l_2$
5. $\forall M.\ \mathsf{nil} \in M \wedge (\forall \mathfrak{a} \in \mathfrak{A}.\ \forall l \in M.\ \mathsf{cons}(\mathfrak{a}, l) \in M)\ \Rightarrow\ \mathbb{L}_{\mathfrak{A}} \subseteq M$

Axiom 5 entails that if $l \in \mathbb{L}_{\mathfrak{A}}$ then $l = \mathsf{nil}$ or $l = \mathsf{cons}(\mathfrak{a}, l')$ for some $\mathfrak{a} \in \mathfrak{A}$ and $l' \in \mathbb{L}_{\mathfrak{A}}$ (to see this specialise M to $\{l \mid l = \mathsf{nil} \vee \exists \mathfrak{a} \in \mathfrak{A}.\, \exists l' \in \mathbb{L}_{\mathfrak{A}}.\, l = \mathsf{cons}(\mathfrak{a}, l')\}$). This and Axiom 4 shows that there are destructors $\mathsf{hd} : \mathbb{L}_{\mathfrak{A}} \nrightarrow \mathfrak{A}$ and $\mathsf{tl} : \mathbb{L}_{\mathfrak{A}} \nrightarrow \mathbb{L}_{\mathfrak{A}}$, which satisfy $\forall l \in \mathbb{L}_{\mathfrak{A}}.\ l = \mathsf{nil} \vee l = \mathsf{cons}(\mathsf{hd}(l), \mathsf{tl}(l))$.

These five Peano-style axioms for lists are equivalent to the single property that if $\mathcal{A}$ is a set, $nl \in \mathcal{A}$ and $cs : \mathfrak{A} \times \mathcal{A} \to \mathcal{A}$, then there is a unique function $f : \mathbb{L}_{\mathfrak{A}} \to \mathcal{A}$ such that $\forall l \in \mathbb{L}_{\mathfrak{A}}.\ f(l) = \texttt{if}\ l = \mathsf{nil}\ \texttt{then}\ nl\ \texttt{else}\ cs(\mathsf{hd}(l), f(\mathsf{tl}(l)))$.

A structure $(\mathcal{A}, nl, cs)$ where $nl \in \mathcal{A}$ and $cs : \mathfrak{A} \times \mathcal{A} \to \mathcal{A}$ is an $\mathfrak{A}$-list algebra. The dual concept is an $\mathfrak{A}$-list coalgebra $(\mathcal{C}, test, dest)$ where $\mathcal{C}$ is the carrier set and $test : \mathcal{C} \nrightarrow \{*\}$ and $dest : \mathcal{C} \nrightarrow \mathfrak{A} \times \mathcal{C}$ are destructors whose domains partition the carrier $\mathcal{C}$.

Some notation is useful when discussing the dual of the $\mathfrak{A}$-list algebra $(\mathbb{L}_{\mathfrak{A}}, \mathsf{nil}, \mathsf{cons})$.

- For any sets X_1, X_2, if $(x_1, x_2) \in X_1 \times X_2$ then $\pi_1(x_1, x_2) = x_1$ and $\pi_2(x_1, x_2) = x_2$. The functions $\pi_1 : X_1 \times X_2 \to X_1$ and $\pi_2 : X_1 \times X_2 \to X_2$ are called projections.
- If $\theta_1 : X_1 \to Y_1$ and $\theta_2 : X_2 \to Y_2$, then $\theta_1 \times \theta_2 : X_1 \times X_2 \to Y_1 \times Y_2$ is defined by $(\theta_1 \times \theta_2)(x_1, x_2) = (\theta_1(x_1), \theta_2(x_2))$. The function $\theta_1 \times \theta_2$ is called the product of functions θ_1 and θ_2. Note that $(\theta_1 \times \theta_2)(x) = (\theta_1(\pi_1(x)), \theta_2(\pi_2(x)))$.
- If E is an expression containing a variable n (e.g. $E = \sigma(n{+}1)$), then $\lambda n.E$ denotes the function that when applied to an argument returns the value obtained by evaluating E after the argument has been substituted for n, so $\lambda n.\, \sigma(n{+}1)$ denotes the function $n \mapsto \sigma(n{+}1)$.
- The identity function is denoted by id, so $\forall x.\, \mathsf{id}(x) = x$ and $\mathsf{id} = \lambda x.\, x$. The notation id_X is used to make the domain of id explicit, so $\mathsf{id}_X : X \to X$.

The dual of the $\mathfrak{A}$-list algebra $(\mathbb{L}_{\mathfrak{A}}, \mathsf{nil}, \mathsf{cons})$ is the $\mathfrak{A}$-list coalgebra $(\overline{\mathbb{L}}_{\mathfrak{A}}, \mathsf{null}, \mathsf{destcons})$ with the property that for any $\mathfrak{A}$-list coalgebra $(\mathcal{C}, test, dest)$, there is exactly one function $g : \mathcal{C} \to \overline{\mathbb{L}}_{\mathfrak{A}}$ such that for all $x \in \mathcal{C}$:

- if $x \in \mathsf{Dom}(test)$ then
  $g(x) \in \mathsf{Dom}(\mathsf{null})$ and $\mathsf{null}(g(x)) = test(x)$;
- if $x \in \mathsf{Dom}(dest)$ then
  $g(x) \in \mathsf{Dom}(\mathsf{destcons})$ and $\mathsf{destcons}(g(x)) = (\mathsf{id} \times g)(dest(x))$.

It turns out that the unique existence of g determines the $\mathfrak{A}$-list coalgebra $(\overline{\mathbb{L}}_{\mathfrak{A}}, \mathsf{null}, \mathsf{destcons})$ to have carrier set $\overline{\mathbb{L}}_{\mathfrak{A}} = \mathbb{L}_{\mathfrak{A}} \cup \mathfrak{A}^{\mathbb{N}}$, where $\mathfrak{A}^{\mathbb{N}}$ is the set of infinite lists of members of $\mathfrak{A}$. Formally $\mathfrak{A}^{\mathbb{N}} = \{\sigma \mid \sigma : \mathbb{N} \to \mathfrak{A}\}$, i.e. infinite lists are represented as functions σ from the natural numbers to $\mathfrak{A}$, with $\sigma(n)$ being the n^{th} element of the list σ, counting from 0, so $\sigma(0)$ is the first element. The nullary destructor $\mathsf{null} : \{\mathsf{nil}\} \to \{*\}$ satisfies $\mathsf{null}(\mathsf{nil}) = *$ and the unary destructor $\mathsf{destcons} : \{l \mid l \neq \mathsf{nil}\} \to \overline{\mathbb{L}}_{\mathfrak{A}}$ is the function that returns the pair $(\mathsf{hd}(l), \mathsf{tl}(l))$

when applied to a non-empty finite list l, and returns the pair $(\sigma(0), \lambda n.\,\sigma(n{+}1))$ when applied to an infinite list σ.

It is useful to extend cons, hd and tl from $\mathbb{L}_\alpha$ to all of $\mathbb{L}_{\mathfrak{A}} \cup \mathfrak{A}^{\mathbb{N}}$ by defining:

$$
\begin{array}{lcl}
\mathsf{cons}(\mathfrak{a},\sigma) & = & \lambda n.\,\texttt{if}\ n=0\ \texttt{then}\ \mathfrak{a}\ \texttt{else}\ \sigma(n{-}1) \\
\mathsf{hd}(\sigma) & = & \sigma(0) \\
\mathsf{tl}(\sigma) & = & \lambda n.\,\sigma(n{+}1))
\end{array}
$$

With these definitions, the unique function $g : \mathcal{C} \to \overline{\mathbb{L}}_{\mathfrak{A}}$ is defined by the single equation:

$$g(x) \ = \ \texttt{if}\ \mathit{test}(x) = *\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}((\mathsf{id} \times g)(\mathit{dest}(x)))$$

which can also be written as:

$$g(x) \ = \ \texttt{if}\ \mathit{test}(x) = *\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}(\pi_1(\mathit{dest}(x)), g(\pi_2(\mathit{dest}(x))))$$

and is equivalent to:

$$\mathsf{null}(g(x)) = \mathit{test}(x) \ \wedge \ \mathsf{hd}(g(x)) = (\pi_1(\mathit{dest}(x)) \ \wedge \ \mathsf{tl}(g(x)) = g(\pi_2(\mathit{dest}(x)))$$

Note the ways in which colists are dual to lists:

- constructors $\mathsf{nil} \in \mathbb{L}_{\mathfrak{A}}$, $\mathsf{cons} : \mathfrak{A} \times \mathbb{L}_{\mathfrak{A}} \to \mathbb{L}_{\mathfrak{A}}$ versus
  destructors $\mathsf{null} : \overline{\mathbb{L}}_{\mathfrak{A}} \ \not\twoheadrightarrow \ \{*\}$, $\mathsf{destcons} : \overline{\mathbb{L}}_{\mathfrak{A}} \ \not\twoheadrightarrow \ \mathfrak{A} \times \overline{\mathbb{L}}_{\mathfrak{A}}$;
- unique $f : \mathbb{L}_{\mathfrak{A}} \to \mathcal{A}$ versus
  unique $g : \mathcal{C} \to \overline{\mathbb{L}}_{\mathfrak{A}}$;
- define f on constructors: $f(\mathsf{nil}) = nl \wedge f(\mathsf{cons}(\mathfrak{a}, l)) = cs(\mathfrak{a}, f(l))$ versus
  define destructors on g: $\mathsf{null}(g(x))=\mathit{test}(x) \wedge \mathsf{destcons}(g(x))=(\mathsf{id}\times g)(\mathit{dest}(x))$;
- recursion $f(l) = \texttt{if}\ l = \mathsf{nil}\ \texttt{then}\ nl\ \texttt{else}\ cs(\mathsf{hd}(l), f(\mathsf{tl}(l)))$ versus
  corecursion $g(x) = \texttt{if}\ \mathit{test}(x) = *\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}((\mathsf{id} \times g)(\mathit{dest}(x)))$.
- recursion *always* terminates versus
  corecursion *sometimes* doesn't terminate (see example below).

Example of corecursion for lists

The Peano coalgebra example used above can be reinterpreted as a $\overline{\mathbb{L}}_{\mathfrak{A}}$ coalgebra.

$$
\begin{array}{ll}
\mathfrak{A} & = \{A, B, C, D, E, F, G, H, I, J\}, \\
\mathcal{C} & = \{A, B, C, D, E, F, G, H, I, J\}, \\
\mathit{test} & = [F \mapsto *, J \mapsto *], \\
\mathit{dest} & = [A \mapsto (A, A), B \mapsto (B, C), C \mapsto (C, B), D \mapsto (D, E), E \mapsto (E, F), \\
& \quad\ G \mapsto (G, H), H \mapsto (H, I), I \mapsto (I, J)].
\end{array}
$$

$\mathsf{Dom}(\mathit{test}) = \{F, J\}$ and $\mathsf{Dom}(\mathit{dest}) = \{A, B, C, D, E, G, H, I\}$, so $\mathsf{Dom}(\mathit{test})$ and $\mathsf{Dom}(\mathit{dest})$ partition $\mathcal{C}$.

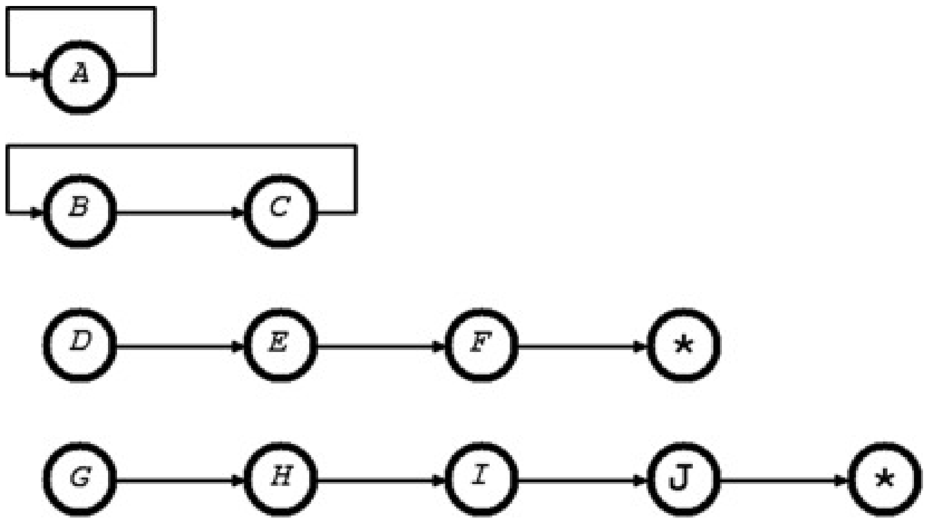

Figure 2:

In the diagram in Figure 1 below: an arrow from x to x' means that if $x \in \mathsf{Dom}(dest)$ then $dest(x) = (x, x')$, and if $x \in \mathsf{Dom}(test)$ then $test(x) = *$.

Consider now the definition of $g : \mathcal{C} \to \overline{\mathbb{L}}_{\mathfrak{A}}$ specified for $x \in \mathcal{C}$ by $g(x) = \texttt{if}\ test(x) = *\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}((\mathsf{id} \times g)(dest(x)))$.

The notation $[\mathfrak{a}_0, \mathfrak{a}_1, \ldots, \mathfrak{a}_n]$ abbreviates $\mathsf{cons}(\mathfrak{a}_0, \mathsf{cons}(\mathfrak{a}_1, \ldots, \mathsf{cons}(\mathfrak{a}_n, \mathsf{nil}) \cdots))$ and $[\,]$ is used as a synonym for nil.

Rewriting $g(A)$ with the equation defining g yields:

$$g(A) = \mathsf{cons}((\mathsf{id} \times g)(dest(A))) = \mathsf{cons}((\mathsf{id} \times g)(A, A)) = \mathsf{cons}(A, g(A)) = [A, g(A)]$$

The only way this can be satisfied is with $g(A)$ being the infinite list of As, i.e. $g(A) = [A, A, A, \ldots]$. Formally this is the function $\lambda n.\, A$.

Rewriting $g(B)$ and $g(C)$ yields:

$$\begin{aligned}
g(B) &= \mathsf{cons}((\mathsf{id} \times g)(dest(B))) \\
&= \mathsf{cons}((\mathsf{id} \times g)(B, C)) \\
&= \mathsf{cons}(B, g(C)) \\
&= \mathsf{cons}(B, \mathsf{cons}((\mathsf{id} \times g)(dest(C)))) \\
&= \mathsf{cons}(B, \mathsf{cons}(C, g(B))) \\
&= [B, C, g(B)] \\
g(C) &= \mathsf{cons}((\mathsf{id} \times g)(dest(C))) \\
&= \mathsf{cons}((\mathsf{id} \times g)(C, B)) \\
&= \mathsf{cons}(C, g(B)) \\
&= \mathsf{cons}(C, \mathsf{cons}((\mathsf{id} \times g)(dest(B)))) \\
&= \mathsf{cons}(C, \mathsf{cons}(B, g(C))) \\
&= [C, B, g(C)]
\end{aligned}$$

The only way these can be satisfied is with $g(B)$ being the infinite list that repeats B followed by C, i.e. $g(B) = [B, C, B, C, B, C, B, C \ldots]$ and $g(C)$ being the infinite list that repeats C followed by B, i.e. $g(C) = [C, B, C, B, C, B, C, B \ldots]$.

Thus if $x \in \{A, B, C\}$, then evaluating $g(x)$ by rewriting with the equation for g returns an infinite list, i.e. a member of $\mathfrak{A}^{\mathbb{N}}$.

On other arguments the rewriting results in a finite list, i.e. a member of $\mathbb{L}_{\mathfrak{A}}$. The rewriting calculations below take bigger steps than the ones above (the expansion of $(\mathsf{id} \times g)(\cdots)$ is omitted).

$$
\begin{aligned}
&g(F) = \mathsf{nil} = [\,]\\
&g(E) = \mathsf{cons}(E, g(F)) = \mathsf{cons}(E, \mathsf{nil}) = [E]\\
&g(D) = \mathsf{cons}(D, g(E)) = \mathsf{cons}(D, [E]) = [D, E]
\end{aligned}
$$

$$
\begin{aligned}
&g(J) = \mathsf{nil} = [\,]\\
&g(I) = \mathsf{cons}(I, g(J)) = \mathsf{cons}(I, \mathsf{nil}) = [I]\\
&g(H) = \mathsf{cons}(H, g(I)) = \mathsf{cons}(H, [I]) = [H, I]\\
&g(G) = \mathsf{cons}(G, g(H)) = \mathsf{cons}(G, [H, I]) = [G, H, I]
\end{aligned}
$$

These rewriting calculations show that:

$$
\begin{aligned}
g = [&A \mapsto [A, A, A, \ldots],\ B \mapsto [B, C, B, C, B, C, B, C, \ldots],\ C \mapsto [C, B, C, B, C, B, C, B, \ldots],\\
&D \mapsto [D, E],\ E \mapsto [E],\ F \mapsto [\,],\ G \mapsto [G, H, I],\ H \mapsto [H, I],\ I \mapsto [I],\ J \mapsto [\,]]
\end{aligned}
$$

For an arbitrary $\mathfrak{A}$-list coalgebra $(\mathcal{C}, \mathit{test}, \mathit{dest})$, the function $g : \mathcal{C} \to \overline{\mathbb{L}}_{\mathfrak{A}}$ satisfying the equation $g(x) = \texttt{if}\ \mathit{test}(x) = *\ \texttt{then nil else}\ \mathsf{cons}((\mathsf{id} \times g)(\mathit{dest}(x)))$ is described by:

$$
g(x) = \begin{cases}
[\mathfrak{a}_0, \ldots, \mathfrak{a}_{n-1}] & \exists x_0 \cdots x_n.\ \ x = x_0 \ \wedge\ \mathit{test}(x_n) = * \ \wedge\ \forall i{<}n.\, \mathit{dest}(x_i) = (\mathfrak{a}_i, x_{i+1})\\
[\mathfrak{a}_0, \mathfrak{a}_1, \mathfrak{a}_2, \ldots] & \exists \sigma : \mathbb{N} \to \mathcal{C}.\ x = \sigma(0) \ \wedge\ \forall i.\, \mathit{dest}(\sigma(i)) = (\mathfrak{a}_i, \sigma(i{+}1))
\end{cases}
$$

The example just given illustrates this.

Coinduction and bisimulation

Induction for natural numbers is the fifth Peano axiom. For lists, it is the fifth of the Peano-like axioms given at the start of the section on Lists and colists above. Both of these are equivalent to the uniqueness of the functions resulting from the principle of recursive definition from $\mathbb{N}$ or $\mathbb{L}_\alpha$ to the carriers of the appropriate algebras.

As far as I can discover there is no canonical notion of coinduction for the dual of natural numbers or lists. The term "coinduction" (actually "co-induction") is generally attributed to Milner and Tofte in their 1991 paper Co-induction in relational semantics[19] whose abstract is:

> An application of the mathematical theory of maximum fixed points of monotonic set operators to relational semantics is presented. It is shown how an important proof method which we call co-induction, a variant of Park's (1969) principle of fixpoint induction, can be used to prove the consistency of the static and the dynamic relational semantics of a small functional programming language with recursive functions.

Milner and Tofte's paper is based on fixed points, but other more recent work on coinduction is based on terminal algebras, which is the approach taken here (see the sections below entitled least and greatest fixed points and Initial and terminal algebras).

The coinduction principle described here is called *bisimulation coinduction*, where a *bisimulation* is a relation R between members of the carrier of a coalgebra. The name "bisimulation coinduction" is not standard, usually just called "coinduction" is used. The principle is derived from the corecursion equation defining the unique functions from arbitrary coalgebras to the conumber coalgebra $\overline{\mathbb{N}}$ or to the colist coalgebra $\overline{\mathbb{L}}_\alpha$. This derivation is given for conumbers and colists in the sections on the justification of bisimulation coinduction for numbers and the justification of bisimulation coinduction for lists.

Bisimulation coinduction for conumbers

For the conatural numbers $\overline{\mathbb{N}}$, a bisimulation is a relation $R \subseteq \overline{\mathbb{N}} \times \overline{\mathbb{N}}$ such that if $(n_1, n_2) \in R$ then either $n_1 = 0$ and $n_2 = 0$ or else $n_1 = \mathsf{S}(n_1')$ and $n_2 = \mathsf{S}(n_2')$ for some $(n_1', n_2') \in R$.

The bisimulation coinduction principle for $\overline{\mathbb{N}}$ states that if R is any bisimulation, then $R \subseteq \{(n, n) \mid n \in \overline{\mathbb{N}}\}$.

Example of bisimulation coinduction for numbers It's hard to come up with examples for $\overline{\mathbb{N}}$ that illustrate useful applications of coinduction. This is because coinduction is actually not much use in practice. The rather contrived example that follows is inspired by part of a (hopefully) more convincing example used later to illustrate coinduction on lists. In the example $\mathsf{even}(n)$ means that n is an even number ($\mathsf{even}(0)$ is considered true) and $\mathsf{odd}(n)$ that n is odd.

Consider the unique function $g : \mathbb{N} \to \overline{\mathbb{N}}$ determined by the Peano coalgebra $(\mathbb{N}, isz, p)$ where $\mathsf{Dom}(isz) = \{n \mid \mathsf{even}(n)\}$ and $\mathsf{Dom}(p) = \{n \mid \mathsf{odd}(n)\}$, and the

[19] http://www.sciencedirect.com/science/article/pii/030439759190033X

destructors are defined by $p(n) = * \Leftrightarrow \mathsf{even}(n)$ and $s(n) = n{+}2$. This function g is defined by:

$g(x) = \texttt{if } isz(x) = * \texttt{ then } 0 \texttt{ else } g(p(x)){+}1$

and with the particular isz and p just specified, is equivalent to:

$g(n) = \texttt{if } \mathsf{even}(n) \texttt{ then } 0 \texttt{ else } g(n{+}2){+}1$

Intuitively this function returns 0 on even numbers and loops on odd numbers, so is equal to $h : \mathbb{N} \to \overline{\mathbb{N}}$ defined by: $h(n) = \texttt{if } \mathsf{even}(n) \texttt{ then } 0 \texttt{ else } \infty$.

This can be proved by showing that $R = \{(g(n), h(n)) \mid n \in \mathbb{N}\}$ is a bisimulation.

Suppose $(n_1, n_2) \in R$, then $n_1 = g(n)$ and $n_2 = h(n)$ for some $n \in \mathbb{N}$.

If n is even then $n_1 = g(n) = 0$ and $n_2 = h(n) = 0$.

If n is not even, then $n_1 = g(n) = g(n{+}2){+}1 = \mathsf{S}(g(n{+}2))$ (including when $g(n{+}2) = \infty$) and $n_2 = h(n) = \infty = \mathsf{S}(\infty) = \mathsf{S}(h(n{+}2))$, as $n{+}2$ is not even if n is not even.

Thus if $(n_1, n_2) \in R$ then either $n_1 = 0$ and $n_2 = 0$ or else $n_1 = \mathsf{S}(n_1')$ and $n_2 = \mathsf{S}(n_2')$ for some $(n_1', n_2') \in R$ – here $n_1' = g(n{+}2)$ and $n_2' = h(n{+}2)$ – so R is a bisimulation, hence by the bisimulation coinduction principle it follows that $\forall n \in \mathbb{N}.\, g(n) = h(n)$.

Justification of bisimulation coinduction for numbers To establish that the uniqueness of corecursively specified functions entails the principle of bisimulation coinduction, let $R \subseteq \overline{\mathbb{N}} \times \overline{\mathbb{N}}$ be a bisimulation. Consider the Peano coalgebra (R, isz_R, p_R), where $isz_R(n_1, n_2) = * \Leftrightarrow n_1 = 0 \wedge n_2 = 0$ and $p_R(n_1, n_2) = (\mathsf{P}(n_1), \mathsf{P}(n_2))$. The definition of a bisimulation ensures that the domains of isz_R and p_R partition the coalgebra carrier set R. By the defining property of $(\overline{\mathbb{N}}, \mathsf{is0}, \mathsf{P})$, there is a unique function $g : R \to \overline{\mathbb{N}}$ such that for all $(n_1, n_2) \in R$:

$g(n_1, n_2) = \texttt{if } isz_R(n_1, n_2) = * \texttt{ then } 0 \texttt{ else } g(p_R(n_1, n_2)){+}1$

i.e. for all $(n_1, n_2) \in R$:

$g(n_1, n_2) = \texttt{if } n_1 = 0 \wedge n_2 = 0 \texttt{ then } 0 \texttt{ else } g(\mathsf{P}(n_1), \mathsf{P}(n_2)){+}1$

Recall the projection functions: $\pi_1(\sigma_1, \sigma_2) = \sigma_1$ and $\pi_2(\sigma_1, \sigma_2) = \sigma_2$. As is about to be shown, it is easy to verify the equation for g is satisfied with both $g = \pi_1$ and $g = \pi_2$.

Taking $g = \pi_1$ and assuming $(n_1, n_2) \in R$:

$\pi_1(n_1, n_2) = \texttt{if } n_1 = 0 \wedge n_2 = 0 \texttt{ then } 0 \texttt{ else } \pi_1(\mathsf{P}(n_1), \mathsf{P}(n_2)){+}1$

which simplifies to:

$n_1 = \texttt{if } n_1 = 0 \wedge n_2 = 0 \texttt{ then } 0 \texttt{ else } \mathsf{P}(n_1){+}1$

which holds for all $(n_1, n_2) \in R$, since if $(n_1, n_2) \in R$ then $n_1 = 0 \Leftrightarrow n_2 = 0$ as R is a bisimulation.

Now take $g = \pi_2$, then assuming $(n_1, n_2) \in R$:

$\pi_2(n_1, n_2) = \texttt{if}\ n_1 = 0 \wedge n_2 = 0\ \texttt{then}\ 0\ \texttt{else}\ \pi_2(\mathsf{P}(n_1), \mathsf{P}(n_2)){+}1$

which simplifies to:

$n_2 = \texttt{if}\ n_1 = 0 \wedge n_2 = 0\ \texttt{then}\ 0\ \texttt{else}\ \mathsf{P}(n_2){+}1$

which also holds for all $(n_1, n_2) \in R$.

Since g is unique, $\pi_1 : R \to \overline{\mathbb{N}}$ and $\pi_2 : R \to \overline{\mathbb{N}}$ must be the same function, so if $(n_1, n_2) \in R$ then $n_1 = \pi_1(n_1, n_2) = \pi_2(n_1, n_2) = n_2$, so $R \subseteq \{(n, n) \mid \in \overline{\mathbb{N}}\}$.

The argument just given shows that the bisimulation coinduction principle follows from the uniqueness of the function $g : \mathcal{C} \to \overline{\mathbb{N}}$ corecursively specified by:

$g(x) = \texttt{if}\ isz(x) = *\ \texttt{then}\ 0\ \texttt{else}\ g(p(x)){+}1$

To prove the reverse implication, i.e. that the bisimulation coinduction principle entails the uniqueness of corecursively specified functions, suppose that:

$$\begin{array}{l} g_1(x) = \texttt{if}\ isz(x) = *\ \texttt{then}\ 0\ \texttt{else}\ g_1(p(x)){+}1 \\ g_2(x) = \texttt{if}\ isz(x) = *\ \texttt{then}\ 0\ \texttt{else}\ g_2(p(x)){+}1 \end{array}$$

then it's easy to see that R, where $R = \{(g_1(x), g_2(x)) \mid x \in \mathcal{C}\}$, is a bisimulation.

If $(n_1, n_2) \in R$ then $n_1 = g_1(x)$ and $n_2 = g_2(x)$ for some $x \in \mathcal{C}$.

If $isz(x) = *$, then $n_1 = g_1(x) = 0$ and $n_2 = g_2(x) = 0$.

If $isz(x) \neq *$, then $n_1 = g_1(x) = g_1(p(x)){+}1 = \mathsf{S}(g_1(p(x)))$ and $n_2 = g_2(x) = g_2(p(x)){+}1 = \mathsf{S}(g_2(p(x)))$. Since $(g_1(p(x)), g_2(p(x))) \in R$ by the definition of R, it follows that R is a bisimulation, hence by the bisimulation coinduction principle: $\forall x \in \mathcal{C}.\, g_1(x) = g_2(x)$.

Bisimulation coinduction for lists

A bisimulation on $\overline{\mathbb{L}}_{\mathfrak{A}}$ is a relation $R \subseteq \overline{\mathbb{L}}_{\mathfrak{A}} \times \overline{\mathbb{L}}_{\mathfrak{A}}$ such that if $(\sigma_1, \sigma_2) \in R$ then either $\sigma_1 = \mathsf{nil}$ and $\sigma_2 = \mathsf{nil}$ or else $\sigma_1 = \mathsf{cons}(\mathfrak{a}, \sigma_1')$ and $\sigma_2 = \mathsf{cons}(\mathfrak{a}, \sigma_2')$ for some $\mathfrak{a} \in \mathfrak{A}$ and $(\sigma_1', \sigma_2') \in R$.

The bisimulation coinduction principle for $\overline{\mathbb{L}}_{\mathfrak{A}}$ states that if R is any bisimulation, then $R \subseteq \{(\sigma, \sigma) \mid \sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}\}$.

Examples of bisimulation coinduction for lists Recall the corecursively defined functions CountFrom, CountUp and CountUpTo from the summary at the beginning of this article.

$$
\begin{array}{lcl}
\mathsf{CountFrom}(n) & = & \mathsf{cons}(n, \mathsf{CountFrom}(n{+}1)) \\
\mathsf{CountUp}(m,n) & = & \texttt{if } m = n \texttt{ then } \mathsf{nil} \texttt{ else } \mathsf{cons}(m, \mathsf{CountUp}(m{+}1, n)) \\
\mathsf{CountUpTo}(m,n) & = & \texttt{if } m \geq n \texttt{ then } \mathsf{nil} \texttt{ else } \mathsf{cons}(m, \mathsf{CountUpTo}(m{+}1, n))
\end{array}
$$

It is asserted in the summary that R_1 and R_2 are bisimulations, where:

$$
\begin{array}{lcl}
R_1 & = & \{(\mathsf{CountUp}(m,n), \mathsf{CountFrom}(m)) \mid m > n\} \\
R_2 & = & \{(\mathsf{CountUp}(m,n), \mathsf{CountUpTo}(m,n)) \mid m \leq n\}
\end{array}
$$

This assertion is verified in the next two paragraphs.

Suppose $(\sigma_1, \sigma_2) \in R_1$, then $\sigma_1 = \mathsf{CountUp}(m,n)$ and $\sigma_2 = \mathsf{CountFrom}(m)$ for some (m,n) with $m > n$. It follows from the definitions of $\mathsf{CountUp}$ and $\mathsf{CountFrom}$ that $\sigma_1 = \mathsf{cons}(m, \mathsf{CountUp}(m{+}1, n))$ and $\sigma_2 = \mathsf{cons}(m, \mathsf{CountFrom}(m{+}1))$. As $m > n$ implies $m{+}1 > n$, it follows that $(\mathsf{CountUp}(m{+}1, n), \mathsf{CountFrom}(m{+}1)) \in R_1$, hence R_1 is a bisimulation, so by bisimulation coinduction: $\forall m\, n \in \mathbb{N}.\, m > n \Rightarrow \mathsf{CountUp}(m,n) = \mathsf{CountFrom}(m)$.

Suppose $(\sigma_1, \sigma_2) \in R_2$, then $\sigma_1 = \mathsf{CountUp}(m,n)$ and $\sigma_2 = \mathsf{CountUpTo}(m,n)$ for some (m,n) with $m \leq n$. If $m = n$ then from the definitions of $\mathsf{CountUp}$ and $\mathsf{CountUpTo}$ it follows that $\sigma_1 = \mathsf{nil}$ and $\sigma_2 = \mathsf{nil}$. If $m < n$, then from the definitions of $\mathsf{CountUp}$ and $\mathsf{CountUpTo}$ it follows that $\sigma_1 = \mathsf{cons}(m, \mathsf{CountUp}(m{+}1, n))$ and $\sigma_2 = \mathsf{cons}(m, \mathsf{CountUpTo}(m{+}1, n))$. As $m < n$ implies $m{+}1 \leq n$, it follows that $(\mathsf{CountUp}(m{+}1, n), \mathsf{CountUpTo}(m{+}1, n)) \in R_2$, hence R_2 is a bisimulation, so by bisimulation coinduction: $\forall m\, n \in \mathbb{N}.\, m \leq n \Rightarrow \mathsf{CountUp}(m,n) = \mathsf{CountUpTo}(m,n)$.

By combining the results of coinduction based on the bisimulations R_1 and R_2, the equation below can be deduced.

$$\mathsf{CountUp}(m,n) = \texttt{if } m \leq n \texttt{ then } \mathsf{CountUpTo}(m,n) \texttt{ else } \mathsf{CountFrom}(m)$$

The remaining examples in this section are adapted from similar ones in *An introduction to (co)algebra and (co)induction*[20] by Bart Jacobs and Jan Rutten, except that here lists can be finite or infinite, but in Jacobs & Rutton they are only infinite. Allowing lists to be finite complicates the examples as there are extra cases to consider. It's not clear whether the examples below illustrate anything significant that isn't already shown by the examples above.

If $\sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}$ – i.e. σ is a finite or infinite list – then $g_{\mathsf{even}}(\sigma)$ is the sublist consisting of those elements at even numbered positions and $g_{\mathsf{odd}}(\sigma)$ is the sublist consisting of those elements at odd numbered positions.

$$
\begin{array}{lcccccccccccccc}
\sigma: & \mathfrak{a}_0 & \mathfrak{a}_1 & \mathfrak{a}_2 & \mathfrak{a}_3 & \mathfrak{a}_4 & \mathfrak{a}_5 & \mathfrak{a}_6 & \mathfrak{a}_7 & \mathfrak{a}_8 & \mathfrak{a}_9 & \mathfrak{a}_{10} & \mathfrak{a}_{11} & \cdots \\
g_{\mathsf{even}}(\sigma): & \mathfrak{a}_0 & & \mathfrak{a}_2 & & \mathfrak{a}_4 & & \mathfrak{a}_6 & & \mathfrak{a}_8 & & \mathfrak{a}_{10} & & \cdots \\
g_{\mathsf{odd}}(\sigma): & & \mathfrak{a}_1 & & \mathfrak{a}_3 & & \mathfrak{a}_5 & & \mathfrak{a}_7 & & \mathfrak{a}_9 & & \mathfrak{a}_{11} & \cdots
\end{array}
$$

The functions g_{even} and g_{odd} are defined by corecursion below, and then two lemmas are proved by bisumulation coinduction:

Lemma 1. $\forall \sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}.\, g_{\mathsf{odd}}(\sigma) = g_{\mathsf{even}}(\mathsf{tl}(\sigma))$

[20] http://homepages.cwi.nl/~janr/papers/files-of-papers/2011_Jacobs_Rutten_new.pdf

Lemma 2. $\forall \sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}.\, \sigma \neq \mathsf{nil} \;\Rightarrow\; \mathsf{tl}(g_{\mathsf{even}}(\sigma)) = g_{\mathsf{odd}}(\mathsf{tl}(\sigma))$

Next $\mathsf{merge}(\sigma_1, \sigma_2)$ is defined by corecursion to be the interleaving of σ_1 and σ_2. Using these two lemmas, it is then shown by bisimulation coinduction that $\mathsf{merge}(g_{\mathsf{even}}(\sigma), g_{\mathsf{odd}}(\sigma)) = \sigma$.

To define g_{even} and g_{odd} consider the following $\mathfrak{A}$-list coalgebras:

$$\begin{array}{l}
\mathbb{C}_{\mathsf{even}} = (\overline{\mathbb{L}}_{\mathfrak{A}},\ \mathsf{test}_{\mathsf{even}},\ \mathsf{dest}_{\mathsf{even}}) \\
\mathbb{C}_{\mathsf{odd}} \;= (\overline{\mathbb{L}}_{\mathfrak{A}},\ \mathsf{test}_{\mathsf{odd}},\ \mathsf{dest}_{\mathsf{odd}})
\end{array}$$

where

$$\begin{array}{lcl}
\mathsf{test}_{\mathsf{even}}(\sigma) = * & \Leftrightarrow & \sigma = \mathsf{nil} \\
\mathsf{test}_{\mathsf{odd}}(\sigma) \;= * & \Leftrightarrow & \sigma = \mathsf{nil} \text{ or } \mathsf{tl}(\sigma) = \mathsf{nil}
\end{array}$$

and

$$\begin{array}{lcl}
\mathsf{dest}_{\mathsf{even}}(\sigma) = & (\mathsf{hd}(\sigma),\ \texttt{if}\ \mathsf{tl}(\sigma) = \mathsf{nil}\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{tl}(\mathsf{tl}(\sigma))) \\
\mathsf{dest}_{\mathsf{odd}}(\sigma) \;= & (\mathsf{hd}(\mathsf{tl}(\sigma)),\ \mathsf{tl}(\mathsf{tl}(\sigma)))
\end{array}$$

The domains of destructors partition $\mathbb{C}_{\mathsf{even}}$, so the definition of $\mathsf{test}_{\mathsf{even}}$ entails that the domain of $\mathsf{dest}_{\mathsf{even}}$ is the set of lists with at least one element. The definitions of $\mathsf{dest}_{\mathsf{even}}$ is illustrated by:

$$\begin{array}{l}
\mathsf{dest}_{\mathsf{even}}([\mathfrak{a}_0]) = (\mathfrak{a}_0, \mathsf{nil}) \\
\mathsf{dest}_{\mathsf{even}}([\mathfrak{a}_0, \mathfrak{a}_1]) = (\mathfrak{a}_0, \mathsf{nil}) \\
\mathsf{dest}_{\mathsf{even}}([\mathfrak{a}_0, \mathfrak{a}_1, \mathfrak{a}_2]) = (\mathfrak{a}_0, [\mathfrak{a}_2]) \\
\mathsf{dest}_{\mathsf{even}}([\mathfrak{a}_0, \mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}_3, \ldots]) = (\mathfrak{a}_0, [\mathfrak{a}_2, \mathfrak{a}_3, \ldots])
\end{array}$$

The domains of destructors partition $\mathbb{C}_{\mathsf{odd}}$, so the definition of $\mathsf{test}_{\mathsf{odd}}$ entails that the domain of $\mathsf{dest}_{\mathsf{odd}}$ is the set of lists with at least two elements. The definition of $\mathsf{dest}_{\mathsf{odd}}$ is illustrated by:

$$\begin{array}{l}
\mathsf{dest}_{\mathsf{odd}}([\mathfrak{a}_0, \mathfrak{a}_1]) = (\mathfrak{a}_1, \mathsf{nil}) \\
\mathsf{dest}_{\mathsf{odd}}([\mathfrak{a}_0, \mathfrak{a}_1, \mathfrak{a}_2]) = (\mathfrak{a}_1, [\mathfrak{a}_2]) \\
\mathsf{dest}_{\mathsf{odd}}([\mathfrak{a}_0, \mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}_3, \ldots]) = (\mathfrak{a}_1, [\mathfrak{a}_2, \mathfrak{a}_3, \ldots])
\end{array}$$

Recall the general form of the unique corecursively specified function

$g : \mathcal{C} \to \overline{\mathbb{L}}_{\mathfrak{A}}$ from the carrier of a coalgebra $(\mathcal{C}, \mathit{test}, \mathit{dest})$ to the carrier of $(\overline{\mathbb{L}}_{\mathfrak{A}}, \mathsf{null}, \mathsf{destcons})$:

$$g(x) \;=\; \texttt{if}\ \mathit{test}(x) = *\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}((\mathsf{id} \times g)(\mathit{dest}(x)))$$

For the coalgebras $\mathbb{C}_{\mathsf{even}}$ and $\mathbb{C}_{\mathsf{odd}}$, this general schema instantiates, respectively, to:

$$\begin{array}{ll}
g_{\mathsf{even}}(\sigma) = & \texttt{if}\ \sigma = \mathsf{nil} \\
& \quad \texttt{then}\ \mathsf{nil} \\
& \quad \texttt{else}\ \mathsf{cons}(\mathsf{hd}(\sigma), g_{\mathsf{even}}(\texttt{if}\ \mathsf{tl}(\sigma) = \mathsf{nil}\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{tl}(\mathsf{tl}(\sigma))))
\end{array}$$

$$\begin{array}{ll}
g_{\mathsf{odd}}(\sigma) = & \texttt{if}\ \sigma = \mathsf{nil} \text{ or } \mathsf{tl}(\sigma) = \mathsf{nil} \\
& \quad \texttt{then}\ \mathsf{nil} \\
& \quad \texttt{else}\ \mathsf{cons}(\mathsf{hd}(\mathsf{tl}(\sigma)), g_{\mathsf{odd}}(\mathsf{tl}(\mathsf{tl}(\sigma))))
\end{array}$$

These equations look intuitively correct.

To prove Lemma 1: $\forall\sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}.\, g_{\mathsf{odd}}(\sigma) = g_{\mathsf{even}}(\mathsf{tl}(\sigma))$, let $R = \{(g_{\mathsf{odd}}(\sigma), g_{\mathsf{even}}(\mathsf{tl}(\sigma))) \mid \sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}\}$. It is shown below that R is a bisimulation.

If $(\sigma_1, \sigma_2) \in R$, then $\sigma_1 = g_{\mathsf{odd}}(\sigma)$ and $\sigma_2 = g_{\mathsf{even}}(\mathsf{tl}(\sigma))$ for some $\sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}$. Three cases need to be considered.

1. If $\sigma = \mathsf{nil}$ or $\mathsf{tl}(\sigma) = \mathsf{nil}$ then $\sigma_1 = \sigma_2 = \mathsf{nil}$ by the definitions of g_{even} and g_{odd}.
2. If $\sigma \neq \mathsf{nil}$ and $\mathsf{tl}(\sigma) \neq \mathsf{nil}$ and $\mathsf{tl}(\mathsf{tl}(\sigma) = \mathsf{nil}$ then
   $\sigma_1 = g_{\mathsf{odd}}(\sigma) = \mathsf{cons}(\mathsf{hd}(\mathsf{tl}(\sigma)), g_{\mathsf{odd}}(\mathsf{nil}))$ and
   $\sigma_2 = g_{\mathsf{even}}(\mathsf{tl}(\sigma)) = \mathsf{cons}(\mathsf{hd}(\mathsf{tl}(\sigma)) g_{\mathsf{even}}(\mathsf{nil}))$.
   As $g_{\mathsf{odd}}(\mathsf{nil}) = \mathsf{nil}$ and $g_{\mathsf{even}}(\mathsf{nil}) = \mathsf{nil}$, $(g_{\mathsf{odd}}(\mathsf{nil}), g_{\mathsf{even}}(\mathsf{nil})) \in R$.
3. If $\sigma \neq \mathsf{nil}$ and $\mathsf{tl}(\sigma) \neq \mathsf{nil}$ and $\mathsf{tl}(\mathsf{tl}(\sigma) \neq \mathsf{nil}$ then
   $\sigma_1 = g_{\mathsf{odd}}(\sigma) = \mathsf{cons}(\mathsf{hd}(\mathsf{tl}(\sigma)), g_{\mathsf{odd}}(\mathsf{tl}(\mathsf{tl}(\sigma))))$ and
   $\sigma_2 = g_{\mathsf{even}}(\mathsf{tl}(\sigma)) = \mathsf{cons}(\mathsf{hd}(\mathsf{tl}(\sigma)), g_{\mathsf{even}}(\mathsf{tl}(\mathsf{tl}(\mathsf{tl}(\sigma)))))$.
   Thus $(\mathsf{tl}(\sigma_1), \mathsf{tl}(\sigma_2)) = (g_{\mathsf{odd}}(\mathsf{tl}(\mathsf{tl}(\sigma))), g_{\mathsf{even}}(\mathsf{tl}(\mathsf{tl}(\mathsf{tl}(\sigma))))) \in R$.

As $(\sigma_1, \sigma_2) \in R$ if either $\sigma_1 = \mathsf{nil}$ and $\sigma_2 = \mathsf{nil}$ or else $\sigma_1 = \mathsf{cons}(\mathfrak{a}, \sigma_1')$ and $\sigma_2 = \mathsf{cons}(\mathfrak{a}, \sigma_2')$ for some $\mathfrak{a} \in \mathfrak{A}$ and $(\sigma_1', \sigma_2') \in R$, it follows that R is a bisimulation.

To prove Lemma 2: $\sigma \neq \mathsf{nil} \Rightarrow \mathsf{tl}(g_{\mathsf{even}}(\sigma)) = g_{\mathsf{odd}}(\mathsf{tl}(\sigma))$,
let $R = \{(\mathsf{tl}(g_{\mathsf{even}}(\sigma)), g_{\mathsf{odd}}(\mathsf{tl}(\sigma)) \mid \sigma \in \overline{\mathbb{L}}_{\mathfrak{A}} \wedge \sigma \neq \mathsf{nil}\}$. It is shown below that R is a bisimulation.

If $(\sigma_1, \sigma_2) \in R$, then $\sigma_1 = \mathsf{tl}(g_{\mathsf{even}}(\sigma))$ and $\sigma_2 = g_{\mathsf{odd}}(\mathsf{tl}(\sigma))$ for some $\sigma \neq \mathsf{nil}$. Three cases need to be considered.

1. If $\mathsf{tl}(\sigma) = \mathsf{nil}$ then $\sigma_1 = \sigma_2 = \mathsf{nil}$ by the definitions of g_{even} and g_{odd}.
2. If $\mathsf{tl}(\sigma) \neq \mathsf{nil}$ and $\mathsf{tl}(\mathsf{tl}(\sigma)) = \mathsf{nil}$ then $\sigma_1 = \mathsf{tl}(g_{\mathsf{even}}(\sigma)) = g_{\mathsf{even}}(\mathsf{tl}(\mathsf{tl}(\sigma))) = \mathsf{nil}$ and $\sigma_2 = g_{\mathsf{odd}}(\mathsf{tl}(\sigma)) = \mathsf{nil}$ by the definitions of g_{even} and g_{odd}.
3. If $\mathsf{tl}(\sigma) \neq \mathsf{nil}$ and $\mathsf{tl}(\sigma) \neq \mathsf{nil}$ and $\mathsf{tl}(\mathsf{tl}(\sigma)) \neq \mathsf{nil}$ then
   $\sigma_1 = \mathsf{tl}(g_{\mathsf{even}}(\sigma)) = g_{\mathsf{even}}(\mathsf{tl}(\mathsf{tl}(\sigma))) = \mathsf{cons}(\mathsf{hd}(\mathsf{tl}(\mathsf{tl}(\sigma))), \mathsf{tl}(g_{\mathsf{even}}(\mathsf{tl}(\mathsf{tl}(\sigma)))))$
   and $\sigma_2 = g_{\mathsf{odd}}(\mathsf{tl}(\sigma)) = \mathsf{cons}(\mathsf{hd}(\mathsf{tl}(\mathsf{tl}(\sigma))), g_{\mathsf{odd}}(\mathsf{tl}(\mathsf{tl}(\mathsf{tl}(\sigma)))))$. As
   $\mathsf{tl}(\mathsf{tl}(\sigma)) \neq \mathsf{nil}$, $(\mathsf{tl}(\sigma_1), \mathsf{tl}(\sigma_2)) = (\mathsf{tl}(g_{\mathsf{even}}(\mathsf{tl}(\mathsf{tl}(\sigma)))), g_{\mathsf{odd}}(\mathsf{tl}(\mathsf{tl}(\mathsf{tl}(\sigma))))) \in R$.

As $(\sigma_1, \sigma_2) \in R$ if either $\sigma_1 = \mathsf{nil}$ and $\sigma_2 = \mathsf{nil}$ or else $\sigma_1 = \mathsf{cons}(\mathfrak{a}, \sigma_1')$ and $\sigma_2 = \mathsf{cons}(\mathfrak{a}, \sigma_2')$ for some $\mathfrak{a} \in \mathfrak{A}$ and $(\sigma_1', \sigma_2') \in R$, it follows that R is a bisimulation.

The function $\mathsf{merge} : \overline{\mathbb{L}}_{\mathfrak{A}} \times \overline{\mathbb{L}}_{\mathfrak{A}} \to \overline{\mathbb{L}}_{\mathfrak{A}}$ interleaves two lists. It will be proved by bisimulation coinduction that $\forall\sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}.\, \mathsf{merge}(g_{\mathsf{even}}(\sigma), g_{\mathsf{odd}}(\sigma)) = \sigma$. Before proving this, the function merge needs to be defined. The natural recursion to achieve this is:

$$
\begin{array}{l}
\mathsf{merge}(\sigma_1, \sigma_2) = \\
\quad \mathtt{if}\ \sigma_1 = \mathsf{nil} \\
\quad\quad \mathtt{then}\ \sigma_2 \\
\quad\quad \mathtt{else} \\
\quad\quad\quad \mathtt{if}\ \mathsf{tl}(\sigma_1) = \mathsf{nil} \\
\quad\quad\quad\quad \mathtt{then}\ \mathsf{cons}(\mathsf{hd}(\sigma_1), \sigma_2) \\
\quad\quad\quad\quad \mathtt{else}\ \mathsf{cons}(\mathsf{hd}(\sigma_1), \mathsf{merge}(\sigma_2, \mathsf{tl}(\sigma_1)))
\end{array}
$$

To make this fit the corecursion format:

$$g(x) \;=\; \mathtt{if}\ test(x) = *\ \mathtt{then}\ \mathsf{nil}\ \mathtt{else}\ \mathsf{cons}((\mathsf{id} \times g)(dest(x)))$$

where, x ranges over pairs (σ_1, σ_2) of lists, the destructors $test$ and $dest$ need to be defined. To achieve this the equation for merge can be reformulated to:

$$
\begin{array}{l}
\mathsf{merge}(\sigma_1, \sigma_2) = \\
\quad \mathtt{if}\ \sigma_1 = \mathsf{nil}\ \text{and}\ \sigma_2 = \mathsf{nil} \\
\quad\quad \mathtt{then}\ \mathsf{nil} \\
\quad\quad \mathtt{else} \\
\quad\quad\quad \mathtt{if}\ \sigma_1 = \mathsf{nil} \\
\quad\quad\quad\quad \mathtt{then}\ \mathsf{cons}(\mathsf{hd}(\sigma_2), \mathsf{merge}(\sigma_1, \mathsf{tl}(\sigma_2)) \\
\quad\quad\quad\quad \mathtt{else}\ \mathsf{cons}(\mathsf{hd}(\sigma_1), \mathsf{merge}(\sigma_2, \mathsf{tl}(\sigma_1)))
\end{array}
$$

This reformulated equation is shown equivalent to the original equation below. The reformulated version becomes an instance of

$$g(x) \;=\; \mathtt{if}\ test(x) = *\ \mathtt{then}\ \mathsf{nil}\ \mathtt{else}\ \mathsf{cons}((\mathsf{id} \times g)(dest(x)))$$

if $g = \mathsf{merge}$ and $test : \{(\mathsf{nil}, \mathsf{nil})\} \to \{*\}$ is (necessarily) defined by:

$$test(\sigma_1, \sigma_2) = * \quad \Leftrightarrow \quad \sigma_1 = \mathsf{nil}\ \text{and}\ \sigma_2 = \mathsf{nil}$$

and $dest : \{(\sigma_1, \sigma_2) \mid \sigma_1 \neq \mathsf{nil}\ \text{or}\ \sigma_2 \neq \mathsf{nil}\} \to \mathfrak{A} \times (\overline{\mathbb{L}}_{\mathfrak{A}} \times \overline{\mathbb{L}}_{\mathfrak{A}})$ by:

$$
\begin{array}{rcl}
dest(\sigma_1, \sigma_2) & = & \mathtt{if}\ \sigma_1 = \mathsf{nil} \\
& & \quad \mathtt{then}\ (\mathsf{hd}(\sigma_2), (\sigma_1, \mathsf{tl}(\sigma_2))) \\
& & \quad \mathtt{else}\ (\mathsf{hd}(\sigma_1), (\sigma_2, \mathsf{tl}(\sigma_1)))
\end{array}
$$

The reformulated equation for merge is then the unique function from the carrier of the coalgebra $(\overline{\mathbb{L}}_{\mathfrak{A}} \times \overline{\mathbb{L}}_{\mathfrak{A}}, test, dest)$ to the carrier of the coalgebra $(\overline{\mathbb{L}}_{\mathfrak{A}}, \mathsf{null}, \mathsf{destcons})$.

The original recursion for merge is equivalent to the reformulated equation because $\sigma_2 = \mathsf{cons}(\mathsf{hd}(\sigma_2), \mathsf{tl}(\sigma_2))$ and $\mathsf{merge}(\mathsf{nil}, \mathsf{tl}(\sigma_2)) = \mathsf{tl}(\sigma_2)$, when $\sigma_2 \neq \mathsf{nil}$. The second of these equations is an instance of $\forall \sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}.\, \mathsf{merge}(\mathsf{nil}, \sigma) = \sigma$, which is proved by bisimulation coinduction by showing that $R = \{(\mathsf{merge}(\mathsf{nil}, \sigma), \sigma) \mid \sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}\}$ is a bisimulation.

To show this let $(\sigma_1, \sigma_2) \in R$, then $\sigma_1 = \mathsf{merge}(\mathsf{nil}, \sigma)$ and $\sigma_2 = \sigma$ for some $\sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}$. If $\sigma = \mathsf{nil}$, then $\sigma_1 = \sigma_2 = \mathsf{nil}$. If $\sigma \neq \mathsf{nil}$ then $\sigma_1 = \mathsf{cons}(\mathsf{hd}(\sigma), \mathsf{merge}(\mathsf{nil}, \mathsf{tl}(\sigma)))$ and $\sigma_2 = \sigma = \mathsf{cons}(\mathsf{hd}(\sigma), \mathsf{tl}(\sigma))$. Thus $(\mathsf{tl}(\sigma_1), \mathsf{tl}(\sigma_2)) = (\mathsf{merge}(\mathsf{nil}, \mathsf{tl}(\sigma)), \mathsf{tl}(\sigma)) \in R$, so R is a bisimulation.

To prove $\mathsf{merge}(g_{\mathsf{even}}(\sigma), g_{\mathsf{odd}}(\sigma)) = \sigma$ for arbitrary $\sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}$, it is sufficient to show that $R = \{(\mathsf{merge}(g_{\mathsf{even}}(\sigma), g_{\mathsf{odd}}(\sigma)), \sigma) \mid \sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}\}$ is a bisimulation.

If $(\sigma_1, \sigma_2) \in R$, then $\sigma_1 = \mathsf{merge}(g_{\mathsf{even}}(\sigma), g_{\mathsf{odd}}(\sigma))$ and $\sigma_2 = \sigma$ for some $\sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}$. Three cases need to be considered.

1. If $\sigma = \mathsf{nil}$ then $\sigma_1 = \mathsf{merge}(\mathsf{nil}, \mathsf{nil}) = \mathsf{nil}$ and $\sigma_2 = \sigma = \mathsf{nil}$.
2. If $\sigma \neq \mathsf{nil}$ and $\mathsf{tl}(\sigma) = \mathsf{nil}$ then $\sigma_1 = \mathsf{merge}(\mathsf{cons}(\mathsf{hd}(\sigma), \mathsf{nil}), \mathsf{nil}) = \mathsf{cons}(\mathsf{hd}(\sigma), \mathsf{nil})$ and $\sigma_2 = \sigma = \mathsf{cons}(\mathsf{hd}(\sigma), \mathsf{nil})$.
3. If $\sigma \neq \mathsf{nil}$ and $\mathsf{tl}(\sigma) \neq \mathsf{nil}$ by the definitions of g_{even}, g_{odd} and merge, and using Lemma 1 and Lemma 2:
$\sigma_1 = \mathsf{merge}(g_{\mathsf{even}}(\sigma), g_{\mathsf{odd}}(\sigma))$
$= \mathsf{merge}(\mathsf{cons}(\mathsf{hd}(\sigma), g_{\mathsf{even}}(\mathsf{tl}(\mathsf{tl}(\sigma)))), \mathsf{cons}(\mathsf{hd}(\mathsf{tl}(\sigma)), g_{\mathsf{odd}}(\mathsf{tl}(\mathsf{tl}(\sigma)))))$
$= \mathsf{cons}(\mathsf{hd}(\sigma), \mathsf{merge}(\mathsf{cons}(\mathsf{hd}(\mathsf{tl}(\sigma)), g_{\mathsf{odd}}(\mathsf{tl}(\mathsf{tl}(\sigma)))), g_{\mathsf{even}}(\mathsf{tl}(\mathsf{tl}(\sigma)))))$
$= \mathsf{cons}(\mathsf{hd}(\sigma), \mathsf{merge}(\mathsf{cons}(\mathsf{hd}(\mathsf{tl}(\sigma)), \mathsf{tl}(g_{\mathsf{even}}(\mathsf{tl}(\sigma)))), g_{\mathsf{odd}}(\mathsf{tl}(\sigma))))$
$= \mathsf{cons}(\mathsf{hd}(\sigma), \mathsf{merge}(\mathsf{cons}(\mathsf{hd}(g_{\mathsf{even}}(\mathsf{tl}(\sigma))), \mathsf{tl}(g_{\mathsf{even}}(\mathsf{tl}(\sigma)))), g_{\mathsf{odd}}(\mathsf{tl}(\sigma))))$
$= \mathsf{cons}(\mathsf{hd}(\sigma), \mathsf{merge}(g_{\mathsf{even}}(\mathsf{tl}(\sigma)), g_{\mathsf{odd}}(\mathsf{tl}(\sigma))))$.
Also $\sigma_2 = \sigma = \mathsf{cons}(\mathsf{hd}(\sigma), \mathsf{tl}(\sigma))$, so
$(\mathsf{tl}(\sigma_1), \mathsf{tl}(\sigma_2)) = (\mathsf{merge}(g_{\mathsf{even}}(\mathsf{tl}(\sigma)), g_{\mathsf{odd}}(\mathsf{tl}(\sigma))), \mathsf{tl}(\sigma)) \in R$.
Hence R is a bisimulation.

Justification of bisimulation coinduction for lists A bisimulation on $\overline{\mathbb{L}}_{\mathfrak{A}}$ is a relation $R \subseteq \overline{\mathbb{L}}_{\mathfrak{A}} \times \overline{\mathbb{L}}_{\mathfrak{A}}$ such that $(\sigma_1, \sigma_2) \in R$ if either $\sigma_1 = \mathsf{nil}$ and $\sigma_2 = \mathsf{nil}$ or else $\sigma_1 \neq \mathsf{nil}$ and $\sigma_2 \neq \mathsf{nil}$ and $\mathsf{hd}(\sigma_1) = \mathsf{hd}(\sigma_2)$ and $(\mathsf{tl}(\sigma_1), \mathsf{tl}(\sigma_2)) \in R$.

To establish that the uniqueness of corecursively specified functions entails the principle of bisimulation coinduction, let $R \subseteq \overline{\mathbb{L}}_{\mathfrak{A}} \times \overline{\mathbb{L}}_{\mathfrak{A}}$ be a bisimulation. Define the $\mathfrak{A}$-list coalgebra $(R, test_R, dest_R)$ by: $test_R(\sigma_1, \sigma_2) = * \Leftrightarrow \sigma_1 = \mathsf{nil} \wedge \sigma_2 = \mathsf{nil}$ and $dest_R(\sigma_1, \sigma_2) = (\mathsf{hd}(\sigma_1), (\mathsf{tl}(\sigma_1), \mathsf{tl}(\sigma_2)))$. Note that if $(\sigma_1, \sigma_2) \in R$ then $\mathsf{hd}(\sigma_1) = \mathsf{hd}(\sigma_2)$, so in the definition of $dest_R$ in the last sentence $\mathsf{hd}(\sigma_1)$ could have been $\mathsf{hd}(\sigma_2)$.

The definition of a bisimulation ensures that the domains of $test_R$ and $dest_R$ partition the coalgebra carrier set R. By the defining property of $(\overline{\mathbb{L}}_{\mathfrak{A}}, \mathsf{null}, \mathsf{destcons})$, there is a unique function $g : R \to \overline{\mathbb{L}}_{\mathfrak{A}}$ such that for all $(\sigma_1, \sigma_2) \in R$:

$$g(\sigma_1, \sigma_2) \;=\; \texttt{if } test_R(\sigma_1, \sigma_2) = * \texttt{ then } \mathsf{nil} \texttt{ else } \mathsf{cons}((\mathsf{id} \times g)(dest_R(\sigma_1, \sigma_2)))$$

i.e. for all $(\sigma_1, \sigma_2) \in R$:

$$g(\sigma_1, \sigma_2) = \texttt{if } \sigma_1 = \mathsf{nil} \wedge \sigma_2 = \mathsf{nil} \texttt{ then } \mathsf{nil} \texttt{ else } \mathsf{cons}(\mathsf{hd}(\sigma_1), g(\mathsf{tl}(\sigma_1), \mathsf{tl}(\sigma_2)))$$

The easy verification that the equation for g is satisfied with both $g = \pi_1$ and $g = \pi_2$ is below.

Take $g = \pi_1$, then:

$$\pi_1(\sigma_1, \sigma_2) = \texttt{if } \sigma_1 = \mathsf{nil} \wedge \sigma_2 = \mathsf{nil} \texttt{ then } \mathsf{nil} \texttt{ else } \mathsf{cons}(\mathsf{hd}(\sigma_1), \pi_1(\mathsf{tl}(\sigma_1), \mathsf{tl}(\sigma_2)))$$

which simplifies to:

$\sigma_1 = \texttt{if}\ \sigma_1 = \mathsf{nil} \wedge \sigma_2 = \mathsf{nil}\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}(\mathsf{hd}(\sigma_1), \mathsf{tl}(\sigma_1))$

which holds since if $(\sigma_1, \sigma_2) \in R$ then $(\sigma_1 = \mathsf{nil}) \Leftrightarrow (\sigma_1 = \mathsf{nil} \wedge \sigma_2 = \mathsf{nil})$ and if $\sigma_1 \neq \mathsf{nil}$ then $\sigma_1 = \mathsf{cons}(\mathsf{hd}(\sigma_1), \mathsf{tl}(\sigma_1))$.

Now take $g = \pi_2$, then:

$\pi_2(\sigma_1, \sigma_2) = \texttt{if}\ \sigma_1 = \mathsf{nil} \wedge \sigma_2 = \mathsf{nil}\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}(\mathsf{hd}(\sigma_1), \pi_2(\mathsf{tl}(\sigma_1), \mathsf{tl}(\sigma_2)))$

which – since if $(\sigma_1, \sigma_2) \in R$ then $\mathsf{hd}(\sigma_1) = \mathsf{hd}(\sigma_2)$ – simplifies to:

$\sigma_2 = \texttt{if}\ \sigma_1 = \mathsf{nil} \wedge \sigma_2 = \mathsf{nil}\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}(\mathsf{hd}(\sigma_2), \mathsf{tl}(\sigma_2))$

which holds since if $(\sigma_1, \sigma_2) \in R$ then $(\sigma_2 = \mathsf{nil}) \Leftrightarrow (\sigma_1 = \mathsf{nil} \wedge \sigma_2 = \mathsf{nil})$ and if $\sigma_2 \neq \mathsf{nil}$ then $\sigma_2 = \mathsf{cons}(\mathsf{hd}(\sigma_2), \mathsf{tl}(\sigma_2))$.

Since g is unique, $\pi_1 : R \to \overline{\mathbb{L}}_{\mathfrak{A}}$ and $\pi_2 : R \to \overline{\mathbb{L}}_{\mathfrak{A}}$ must be the same function, so if $(\sigma_1, \sigma_2) \in R$ then $\sigma_1 = \pi_1(\sigma_1, \sigma_2) = \pi_2(\sigma_1, \sigma_2) = \sigma_2$. Thus $R \subseteq \{(\sigma, \sigma) \mid \sigma \in \overline{\mathbb{L}}_{\mathfrak{A}}$.

The argument just given shows that the bisimulation coinduction principle follows from the uniqueness of the function $g : \mathcal{C} \to \overline{\mathbb{L}}_{\mathfrak{A}}$ corecursively specified by:

$g(x) \ = \ \texttt{if}\ \mathit{test}(x) = *\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}((\mathsf{id} \times g)(\mathit{dest}(x)))$

To prove the implication in the other direction, i.e. that the bisimulation coinduction principle entails the uniqueness of corecursively specified functions, suppose that:

$$\begin{array}{lcl} g_1(x) & = & \texttt{if}\ \mathit{test}(x) = *\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}((\mathsf{id} \times g_1)(\mathit{dest}(x))) \\ g_2(x) & = & \texttt{if}\ \mathit{test}(x) = *\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}((\mathsf{id} \times g_2)(\mathit{dest}(x))) \end{array}$$

then it's easy to see that R, where $R = \{(g_1(x), g_2(x)) \mid x \in \mathcal{C}\}$, is a bisimulation. Here is the argument.

If $(\sigma_1, \sigma_2) \in R$ then $\sigma_1 = g_1(x)$ and $\sigma_2 = g_2(x)$ for some $x \in \mathcal{C}$.

If $\mathit{test}(x) = *$, then $\sigma_1 = g_1(x) = \mathsf{nil}$ and $\sigma_2 = g_2(x) = \mathsf{nil}$.

If $\mathit{test}(x) \neq *$,
then $\sigma_1 = g_1(x) = \mathsf{cons}((\mathsf{id} \times g_1)(\mathit{dest}(x))) = \mathsf{cons}(\pi_1(\mathit{dest}(x)), g_1(\pi_2(\mathit{dest}(x))))$
and $\sigma_2 = g_2(x) = \mathsf{cons}((\mathsf{id} \times g_2)(\mathit{dest}(x))) = \mathsf{cons}(\pi_1(\mathit{dest}(x)), g_2(\pi_2(\mathit{dest}(x))))$,
so as $(g_1(\pi_2(\mathit{dest}(x))), g_2(\pi_2(\mathit{dest}(x)))) \in R$, it follows that R is a bisimulation and hence by bisimulation coinduction $\forall x \in \mathcal{C}.\, g_1(x) = g_2(x)$.4

$\mathbb{F}$-algebras and $\mathbb{F}$-coalgebras

$\mathbb{F}$-algebras and $\mathbb{F}$-coalgebras are a uniform framework with numbers and lists being special cases. *An introduction to (co)algebra and (co)induction*[21] by Bart

[21] http://homepages.cwi.nl/~janr/papers/files-of-papers/2011_Jacobs_Rutten_new.pdf

Jacobs and Jan Rutten is a great tutorial, so only an outline of some of the core ideas is given here – just enough to explain how the particular number and list algebras and coalgebras described above fit into the framework.

The $\mathbb{F}$ in $\mathbb{F}$-algebras and $\mathbb{F}$-coalgebras is an operator that maps a set X, the algebra or coalgebra carrier, to a disjoint union of sets that represents the arities of the operators – constructors for algebras and destructors for coalgebras.

Set theory notation

The examples of $\mathbb{F}$ below use the following set theory concepts and notation.

- If $\theta_1 : X \to Y$ and $\theta_2 : Y \to Z$, then $\theta_2 \circ \theta_1 : X \to Z$ is the function composition defined by $\forall x \in X.\,(\theta_2 \circ \theta_1)(x) = \theta_2(\theta_1(x))$. Note that $\mathsf{id}_Y \circ \theta_1 = \theta_1$ and $\theta_1 \circ \mathsf{id}_X = \theta_1$.
- The disjoint union of sets X and Y is just the union $X \cup Y$ when X and Y are disjoint. This is the case for the examples here. If X and Y have elements in common, then they are 'forced' to be disjoint … but details of how this is done are not needed here.
- $X + Y$ denotes the disjoint union of X and Y. If $x \in X + Y$ then either $x \in X$ or $x \in Y$, but not both. $X + Y$ is sometimes called the sum of X and Y.
- If $\theta_1 : X_1 \to Y_1$ and $\theta_2 : X_2 \to Y_2$, then $\theta_1 + \theta_2 : X_1 + X_2 \to Y_1 + Y_2$ is defined by: $(\theta_1 + \theta_2)(x) = \texttt{if}\ x \in X_1\ \texttt{then}\ \theta_1(x)\ \texttt{else}\ \theta_2(x)$.
- $\mathbf{1}$ denotes the single-element set $\{*\}$. It's assumed that $*$ isn't a member of any of the carrier sets in the examples, so that the sums in the definitions of $\mathbb{F}_{\mathbb{N}}$ and $\mathbb{F}_{\mathbb{L}_{\mathfrak{A}}}$ below are between disjoint sets.

Numbers and lists

The particular $\mathbb{F}$s for natural numbers and lists are $\mathbb{F}_{\mathbb{N}}$ and $\mathbb{F}_{\mathbb{L}_{\mathfrak{A}}}$, defined by:

$$\begin{aligned}\mathbb{F}_{\mathbb{N}}(X) &= \mathbf{1} + X\\ \mathbb{F}_{\mathbb{L}_{\mathfrak{A}}}(X) &= \mathbf{1} + (\mathfrak{A} \times X)\end{aligned}$$

The $\mathbb{F}$ for lists that are only infinite is:

$$\mathbb{F}_{\mathfrak{A}^{\mathbb{N}}}(X) = \mathfrak{A} \times X$$

In general, $\mathbb{F}(X)$ is a 'polynomial' built out of X and other sets (e.g. $\mathbf{1}$ and $\mathfrak{A}$ in the examples above) using disjoint sum and Cartesian product.

An $\mathbb{F}$-algebra is a pair $(\mathcal{A}, a)$ where $a : \mathbb{F}(\mathcal{A}) \to \mathcal{A}$.

The $\mathbb{F}_{\mathbb{N}}$-algebra $a : \mathbb{F}_{\mathbb{N}}(\mathcal{A}) \to \mathcal{A}$ represents the Peano algebra $(\mathcal{A}, z, s)$ where a is the function defined by: $a(*) = z$ and $a(x) = s(x)$ when $x \in \mathcal{A}$.

The $\mathbb{F}_{\mathbb{L}_{\mathfrak{A}}}$-algebra $a : \mathbb{F}_{\mathbb{L}_{\mathfrak{A}}}(\mathcal{A}) \to \mathcal{A}$ represents the $\mathfrak{A}$-list algebra $(\mathcal{A}, nl, cs)$ where a is the function defined by: $a(*) = nl$ and $a(\mathfrak{a}, x) = cs(\mathfrak{a}, x)$ when $(\mathfrak{a}, x) \in \mathfrak{A} \times \mathcal{A}$.

Note how the nullary operators – the distinguished elements z and nl – are represented as function from $\mathbf{1}$ to the carrier.

An $\mathbb{F}$-colgebra is a pair $(\mathcal{C}, c)$ where $c : \mathcal{C} \to \mathbb{F}(\mathcal{C})$.

The $\mathbb{F}_{\mathbb{N}}$-coalgebra $c : \mathcal{C} \to \mathbb{F}_{\mathbb{N}}(\mathcal{C})$ represents the Peano coalgebra $(\mathcal{C}, isz, p)$ where c is the function defined by: $c(x) = *$ when $x \in \mathsf{Dom}(isz)$ – i.e. $isz(x) = *$ – and $c(x) = p(x)$ when $x \in \mathsf{Dom}(p)$. This works because the domains of the destructors isz and p partition the carrier set $\mathcal{C}$.

The $\mathbb{F}_{\mathbb{L}_{\mathfrak{A}}}$-coalgebra $c : \mathcal{C} \to \mathbb{F}_{\mathbb{L}_{\mathfrak{A}}}(\mathcal{C})$ represents the $\mathfrak{A}$-list algebra $(\mathcal{C}, test, dest)$ where c is the function defined by: $c(x) = *$ when $x \in \mathsf{Dom}(test)$ – i.e. $test(x) = *$ – and $c(x) = dest(x)$ when $x \in \mathsf{Dom}(dest)$. This works because the domains of the destructors $test$ and $dest$ partition the carrier set $\mathcal{C}$.

Notice the duality: algebra $a : \mathbb{F}(\mathcal{A}) \to \mathcal{A}$ versus coalgebra $c : \mathcal{C} \to \mathbb{F}(\mathcal{C})$.

Morphisms

The concept of a *morphism* is needed to generalise the unique function properties characterising conumbers and colists. To say what a morphism is, $\mathbb{F}$ operators are defined on functions as well as on sets.

If $\theta : X \to Y$ then $\mathbb{F}(\theta) : \mathbb{F}(X) \to \mathbb{F}(Y)$ is the natural extension of θ as illustrated by the following examples.

- If $X = Y = \mathbf{1}$ then necessarily $\theta = \mathsf{id}_{\mathbf{1}}$ and then $\mathbb{F}(\theta) = \mathbb{F}(\mathsf{id}_{\mathbf{1}}) = \mathsf{id}_{\mathbf{1}}$.
- If $\mathbb{F}(X) = \mathbf{1} + X$ then $\mathbb{F}(\theta) : \mathbf{1} + X \to \mathbf{1} + Y$ and $\mathbb{F}(\theta) = \mathsf{id}_{\mathbf{1}} + \theta$.
- If $\mathbb{F}(X) = \mathbf{1} + (\mathfrak{A} \times X)$ then $\mathbb{F}(\theta) : \mathbf{1} + (\mathfrak{A} \times X) \to \mathbf{1} + (\mathfrak{A} \times Y)$ and $\mathbb{F}(\theta) = \mathsf{id}_{\mathbf{1}} + (\mathsf{id}_{\mathfrak{A}} \times \theta)$.

A function $f : \mathcal{A}_1 \to \mathcal{A}_2$ is a morphism from an $\mathbb{F}$-algebra $(\mathcal{A}_1, a_1)$ to an $\mathbb{F}$-algebra $(\mathcal{A}_2, a_2)$ if and only if $f \circ a_1 = a_2 \circ \mathbb{F}(f)$.

To illustrate the definition of an $\mathbb{F}$-algebra morphism, consider an $\mathbb{F}_{\mathbb{N}}$-algebra morphism $f : \mathcal{A}_1 \to \mathcal{A}_2$ from an $\mathbb{F}_{\mathbb{N}}$-algebra $(\mathcal{A}_1, a_1)$ corresponding to the Peano algebra $(\mathcal{A}_1, z_1, s_1)$ to an $\mathbb{F}_{\mathbb{N}}$-algebra $(\mathcal{A}_2, a_2)$ corresponding to the Peano algebra $(\mathcal{A}_2, z_2, s_2)$.

The condition for f to be an $\mathbb{F}_{\mathbb{N}}$-algebra morphism is $f \circ a_1 = a_2 \circ \mathbb{F}_{\mathbb{N}}(f)$.

Expanding the definition of $\mathbb{F}_{\mathbb{N}}(f)$ converts this equation to $f \circ a_1 = a_2 \circ (\mathsf{id}_{\mathbf{1}} + f)$, which means $\forall x \in (\mathbf{1} + \mathcal{A}_1).\ f(a_1(x)) = a_2((\mathsf{id}_{\mathbf{1}} + f)(x))$.

Now, if $x \in (\mathbf{1} + \mathcal{A}_1)$ then either $x \in \mathbf{1}$ or $x \in \mathcal{A}_1$, so there are two cases to consider.

i. If $x \in \mathbf{1}$ then $x = *$ and $a_1(x) = z_1$ so $f(a_1(x)) = f(z_1)$. If $x = *$ then $(\mathsf{id}_{\mathbf{1}} + f)(x) = \mathsf{id}_{\mathbf{1}}(x) = x = *$, so $a_2((\mathsf{id}_{\mathbf{1}} + f)(x)) = a_2(*) = z_2$. Thus the equation $f(a_1(x)) = a_2((\mathsf{id}_{\mathbf{1}} + f)(x))$ reduces to $f(z_1) = z_2$.

ii. If $x \in \mathcal{A}_1$ then $a_1(x) = s_1(x)$, so $f(a_1(x)) = f(s_1(x))$. If $x \in \mathcal{A}_1$ then $(\mathsf{id}_{\mathbf{1}} + f)(x) = f(x)$, so $a_2((\mathsf{id}_{\mathbf{1}} + f)(x)) = a_2(f(x)) = s_2(f(x))$.
Thus the equation $f(a_1(x)) = a_2((\mathsf{id}_{\mathbf{1}} + f)(x))$ reduces to $f(s_1(x)) = s_2(f(x))$.

A morphism from the $\mathbb{F}_{\mathbb{N}}$-algebra corresponding to the natural numbers $(\mathbb{N}, 0, \mathsf{S})$ to an $\mathbb{F}_{\mathbb{N}}$-algebra corresponding to $(\mathcal{A}, z, s)$ is a function $f : \mathbb{N} \to \mathcal{A}$ such that $f(0) = z$ and $\forall n \in \mathbb{N}.\, f(\mathsf{S}(n)) = s(f(n))$, i.e.: $f(n) = \texttt{if}\ n{=}0\ \texttt{then}\ z\ \texttt{else}\ s(f(n{-}1))$. This is the recursive equation characterising the natural numbers.

A function $g : \mathcal{C}_1 \to \mathcal{C}_2$ is a morphism from an $\mathbb{F}$-coalgebra $(\mathcal{C}_1, c_1)$ to an $\mathbb{F}$-coalgebra $(\mathcal{C}_2, c_2)$ if and only if $c_2 \circ g = \mathbb{F}(g) \circ c_1$.

To see what morphisms between coalgebras are, consider $\mathbb{F}_{\mathbb{N}}$-coalgebras. Recall: $\mathbb{F}_{\mathbb{N}}(X) = \mathbf{1} + X$.

Suppose $g : \mathcal{C}_1 \to \mathcal{C}_2$ is a morphism from an $\mathbb{F}_{\mathbb{N}}$-coalgebra $c_1 : \mathcal{C}_1 \to \mathbb{F}_{\mathbb{N}}(\mathcal{C}_1)$ corresponding to a Peano coalgebra $(\mathcal{C}_1, isz_1, p_1)$ to an $\mathbb{F}_{\mathbb{N}}$-coalgebra $c_2 : \mathcal{C}_2 \to \mathbb{F}_{\mathbb{N}}(\mathcal{C}_2)$ corresponding to a Peano coalgebra $(\mathcal{C}_2, isz_2, p_2)$.

If $g : \mathcal{C}_1 \to \mathcal{C}_2$ is a morphism then $c_2(g(x)) = \mathbb{F}_{\mathbb{N}}(g)(c_1(x))$ holds. As $\mathbb{F}_{\mathbb{N}}(g) = \mathsf{id}_{\mathbf{1}} + g$, the right hand side of this morphism equation can be simplified corresponding to the two cases above:

- if $x \in \mathsf{Dom}(isz_1)$ then $\mathbb{F}_{\mathbb{N}}(g)(c_1(x)) = (\mathsf{id}_{\mathbf{1}} + g)(*) = \mathsf{id}_{\mathbf{1}}(*) = *$;
- if $x \in \mathsf{Dom}(p_1)$ then $\mathbb{F}_{\mathbb{N}}(g)(c_1(x)) = (\mathsf{id}_{\mathbf{1}} + g)(p_1(x)) = g(p_1(x))$;

so for these two cases the morphism equation $c_2(g(x)) = \mathbb{F}_{\mathbb{N}}(g)(c_1(x))$ simplifies to:

- if $x \in \mathsf{Dom}(isz_1)$ then $c_2(g(x)) = *$;
- if $x \in \mathsf{Dom}(p_1)$ then $c_2(g(x)) = g(p_1(x))$.

Now $c_2(g(x)) = *$ if and only if $isz_2(g(x)) = *$ and $c_2(g(x)) = g(p_1(x))$ if and only if $g(x) \notin \mathsf{Dom}(isz_2)$ and hence $g(x) \in \mathsf{Dom}(p_2)$ and then $c_2(g(x)) = p_2(g(x))$, so the two cases of the morphism equation further simplify to:

- if $x \in \mathsf{Dom}(isz_1)$ then $isz_2(g(x)) = *$;

- if $x \in \mathsf{Dom}(p_1)$ then $p_2(g(x)) = g(p_1(x))$.

Thus a morphism from an $\mathbb{F}_{\mathbb{N}}$-coalgebra corresponding to $(\mathcal{C}, isz, p)$ to the $\mathbb{F}_{\mathbb{N}}$-coalgebra corresponding to the conatural numbers $(\overline{\mathbb{N}}, \mathsf{is0}, \mathsf{P})$ is a function $g : \mathcal{C} \to \overline{\mathbb{N}}$ such that:

- if $isz(x) = *$ then $g(x) = 0$;
- if $x \in \mathsf{Dom}(p)$ then $\mathsf{P}(g(x)) = g(p(x))$.

These two conditions are equivalent to g satisfying the equation:

$g(x) = \texttt{if } isz(x) = * \texttt{ then } 0 \texttt{ else } g(p(x))+1$.

This is the corecursion equation characterising the conatural numbers.

Initial and terminal algebras

An *initial* $\mathbb{F}$-algebra is one for which there is a unique morphism from it to any other $\mathbb{F}$-algebra. The natural numbers are characterised as being the unique initial $\mathbb{F}_{\mathbb{N}}$-algebra and the finite lists of members of $\mathfrak{A}$ are characterised as being the unique initial $\mathbb{F}_{\mathbb{L}_{\mathfrak{A}}}$-algebra.

A *terminal* $\mathbb{F}$-coalgebra is one for which there is a unique morphism to it from any other $\mathbb{F}$-coalgebra. Terminal coalgebras are sometimes called *final* coalgebras. The conatural numbers are characterised as being the unique terminal $\mathbb{F}_{\mathbb{N}}$-coalgebra. and the finite and infinite lists of elements of $\mathfrak{A}$ are characterised as being the unique terminal $\mathbb{F}_{\mathbb{L}_{\mathfrak{A}}}$-coalgebra.

Least and greatest fixed points

Some presentations of induction and coinduction are based around least and greatest fixed points (e.g. *A Tutorial on Co-induction and Functional Programming*[22] by A.D. Gordon) whilst others are based on initial and terminal algebras (e.g. *An introduction to (co)algebra and (co)induction*[23] by Jacobs & Rutten). The algebra-coalgebra view has been taken here, but in this section its relation to fixed points is superficially sketched.

There are at least two ways that fixed points arise. The first is to provide a uniform way to construct initial $\mathbb{F}$-algebras and terminal $\mathbb{F}$-coalgebras for a wide class of $\mathbb{F}$s. Very roughly, the idea is that a least fixed point of $\mathbb{F}$ yields an initial $\mathbb{F}$-algebra and a greatest fixed point yields a terminal $\mathbb{F}$-coalgebra.

[22] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.8.7706&rep=rep1&type=pdf
[23] http://homepages.cwi.nl/~janr/papers/files-of-papers/2011_Jacobs_Rutten_new.pdf

These algebras and coalgebras can be explicitly constructed using a generalisation (dualised for coalgebras) of the proof of the Tarski-Knaster fixed point theorem. The mathematics of this generalisation is way beyond my comfort zone and I don't attempt to explain it here (further discussion can be found in Section 14 starting on Page 55 of Rutten's paper Universal coalgebra: a theory of systems[24]).

Fixed points, numbers and conumbers

A hint of the idea behind the first way fixed points arise can be glimpsed by looking at numbers. If the mapping $\mathcal{F}_{\mathbb{N}}$ from subsets of $\overline{\mathbb{N}}$ to subsets of $\overline{\mathbb{N}}$ is defined by:

$$\mathcal{F}_{\mathbb{N}}(X) = \{0\} \cup \{\mathsf{S}(x) \mid x \in X\}$$

then $\mathbb{N}$ and $\overline{\mathbb{N}}$ are both fixed points of $\mathcal{F}_{\mathbb{N}}$ – $\mathcal{F}_{\mathbb{N}}(\mathbb{N}) = \mathbb{N}$ and $\mathcal{F}_{\overline{\mathbb{N}}}(\overline{\mathbb{N}}) = \overline{\mathbb{N}}$ – but $\mathbb{N}$ is the least fixed point and $\overline{\mathbb{N}}$ is the greatest fixed point.

$\mathbb{N}$ is also the least pre-fixed point of $\mathcal{F}_{\mathbb{N}}$, that is the least X such that $\mathcal{F}_{\mathbb{N}}(X) \subseteq X$ and hence $\mathbb{N} = \bigcap\{X \mid \mathcal{F}_{\mathbb{N}}(X) \subseteq X\}$.

Dually $\overline{\mathbb{N}}$ is the greatest post-fixed point of $\mathcal{F}_{\mathbb{N}}$, that is the greatest X such that $X \subseteq \mathcal{F}_{\mathbb{N}}(X)$ and hence $\overline{\mathbb{N}} = \bigcup\{X \mid X \subseteq \mathcal{F}_{\mathbb{N}}(X)\}$.

The second way fixed points arise is as a justification of induction and coinduction proof principles.

To illustrate this, compare proving $\forall n \in \mathbb{N}.\, \theta_1(n) = \theta_2(n)$ by induction, where $\theta_1 : \mathbb{N} \to \mathcal{A}$ and $\theta_2 : \mathbb{N} \to \mathcal{A}$, with proving $\forall x \in \mathcal{C}.\, \phi_1(x) = \phi_2(x)$ by coinduction, where $\phi_1 : \mathcal{C} \to \overline{\mathbb{N}}$ and $\phi_2 : \mathcal{C} \to \overline{\mathbb{N}}$.

To prove $\forall n \in \mathbb{N}.\, \theta_1(n) = \theta_2(n)$ by induction, let $P = \{n \mid \theta_1(n) = \theta_2(n)\}$, then the proof of $\forall n \in \mathbb{N}.\, n \in P$ by induction on n consists of the base case $0 \in P$ and the induction step $\forall n.\, n \in P \;\Rightarrow\; \mathsf{S}(n) \in P$.

This induction argument can be seen as an application of least fixed points because the base and induction correspond to proving that $\mathcal{F}_{\mathbb{N}}(P) \subseteq P$, i.e. that P is a pre-fixed point of $\mathcal{F}_{\mathbb{N}}$, so as $\mathbb{N}$ is the least pre-fixed point of $\mathcal{F}_{\mathbb{N}}$ it follows that $\mathbb{N} \subseteq P$, hence $\forall n \in \mathbb{N}.\, n \in P$.

To prove $\forall x \in \mathcal{C}.\, \phi_1(x) = \phi_2(x)$ by coinduction, let $R = \{(\phi_1(x), \phi_2(x)) \mid x \in \mathcal{C}\}$, then the proof by coinduction consists in proving R is a bisimulation, i.e. that for all x: either $\phi_1(x) = 0$ and $\phi_2(x) = 0$ or else $\phi_1(x) = \mathsf{S}(\phi_1(x'))$ and $\phi_2(x) = \mathsf{S}(\phi_2(x'))$, for some $x' \in \mathcal{C}$.

This coinductive argument can be seen as an application of greatest fixed points because the set of pairs $\mathsf{EQ}_{\overline{\mathbb{N}}} = \{(n, n) \mid n \in \overline{\mathbb{N}}\}$ is the greatest post-fixed point of $\mathcal{B}_{\overline{\mathbb{N}}}$, where $\mathcal{B}_{\overline{\mathbb{N}}}$ maps subsets $\overline{\mathbb{N}} \times \overline{\mathbb{N}}$ to subsets of $\overline{\mathbb{N}} \times \overline{\mathbb{N}}$ and is defined by:

[24] https://fldit-www.cs.uni-dortmund.de/~peter/Rutten/UniversalCoalgebra.pdf

$\mathcal{B}_{\overline{\mathbb{N}}}(R) = \{(0,0)\} \cup \{(\mathsf{S}(n_1), \mathsf{S}(n_2)) \mid (n_1, n_2) \in R\}$

$\mathsf{EQ}_{\overline{\mathbb{N}}}$ being a post-fixed point means $\mathsf{EQ}_{\overline{\mathbb{N}}} \subseteq \mathcal{B}_{\overline{\mathbb{N}}}(\mathsf{EQ}_{\overline{\mathbb{N}}})$ and being the greatest post-fixed point means that if $R \subseteq \mathcal{B}_{\overline{\mathbb{N}}}(R)$ then $R \subset \mathsf{EQ}_{\overline{\mathbb{N}}}$.

Another way of saying that $\mathsf{EQ}_{\overline{\mathbb{N}}}$ is the greatest post-fixed point of $\mathcal{B}_{\overline{\mathbb{N}}}$ is with the equation $\mathsf{EQ}_{\overline{\mathbb{N}}} = \bigcup\{R \mid R \subseteq \mathcal{B}_{\overline{\mathbb{N}}}(R)\}$.

R is a bisimulation if $R \subseteq \mathcal{B}_{\overline{\mathbb{N}}}(R)$. The principle of bisimulation coinduction is that if R is a bisimulation then $R \subseteq \mathsf{EQ}_{\overline{\mathbb{N}}}$, i.e. if $R is a bisimulation then $\forall n_1\, n_2 \in \overline{\mathbb{N}}.\, (n_1, n_2) \in R \;\Rightarrow\; n_1 = n_2$.

Summary of fixed point proof rules for numbers and conumbers

If $\theta_1, \theta_2 : \mathbb{N} \to \mathcal{A}$ then $\forall n \in \mathbb{N}.\, \theta_1(n) = \theta_2(n)$ is proved by induction using the rule:

if $P = \{n \mid \theta_1(n) = \theta_2(n)\}$ and $\mathcal{F}_{\mathbb{N}}(P) \subseteq P$ then $\mathbb{N} \subseteq P$

This expands to:

$$\begin{array}{l} \theta_1(0) = \theta_2(0) \wedge (\forall n \in \mathbb{N}.\, \theta_1(n) = \theta_2(n) \;\Rightarrow\; \theta_1(\mathsf{S}(n)) = \theta_2(\mathsf{S}(n))) \\ \;\Rightarrow\; \forall n \in \mathbb{N}.\, \theta_1(n) = \theta_2(n) \end{array}$$

If $\phi_1, \phi_2 : \mathcal{C} \to \overline{\mathbb{N}}$ then $\forall x \in \mathcal{C}.\, \phi_1(x) = \phi_2(x)$ is proved by coinduction using:

if $R = \{(\phi_1(x), \phi_2(x)) \mid x \in \mathcal{C}\}$ and $R \subseteq \mathcal{B}_{\overline{\mathbb{N}}}(R)$ then $R \subseteq \mathsf{EQ}_{\overline{\mathbb{N}}}$

This expands to:

$$\begin{array}{l} (\forall x \in \mathcal{C}. \\ \quad (\phi_1(x) = 0 \wedge \phi_2(x) = 0) \;\vee\; \exists x'.\, \phi_1(x) = \mathsf{S}(\phi_1(x')) \wedge \phi_2(x) = \mathsf{S}(\phi_2(x'))) \\ \;\Rightarrow\; \forall x \in \mathcal{C}.\, \phi_1(x) = \phi_2(x) \end{array}$$

Fixed points, lists and colists

The details of how fixed points relate to lists and colists are analogous to numbers, so are only briefly summarised here.

If the mapping $\mathcal{F}_{\mathbb{L}_{\mathfrak{A}}}$ from subsets of $\overline{\mathbb{L}}_{\mathfrak{A}}$ to subsets of $\overline{\mathbb{L}}_{\mathfrak{A}}$ is defined by:

$\mathcal{F}_{\mathbb{L}_{\mathfrak{A}}}(X) = \{\mathsf{nil}\} \cup \{\mathsf{cons}(\mathfrak{a}, x) \mid (\mathfrak{a} \in \mathfrak{A} \wedge x \in X\}$

then $\mathbb{L}_{\mathfrak{A}}$ and $\overline{\mathbb{L}}_{\mathfrak{A}}$ are both fixed points of $\mathcal{F}_{\mathbb{L}_{\mathfrak{A}}}$, but $\mathbb{L}_{\mathfrak{A}}$ is the least fixed point and $\overline{\mathbb{L}}_{\mathfrak{A}}$ is the greatest fixed point.

$\mathbb{L}_{\mathfrak{A}}$ is also the least pre-fixed point of $\mathcal{F}_{\mathbb{L}_{\mathfrak{A}}}$, that is the least X such that $\mathcal{F}_{\mathbb{L}_{\mathfrak{A}}}(X) \subseteq X$ and hence $\mathbb{L}_{\mathfrak{A}} = \bigcap\{X \mid \mathcal{F}_{\mathbb{L}_{\mathfrak{A}}}(X) \subseteq X\}$.

Dually $\overline{\mathbb{L}}_{\mathfrak{A}}$ is the greatest post-fixed point of $\mathcal{F}_{\mathbb{L}_{\mathfrak{A}}}$, that is the greatest X such that $X \subseteq \mathcal{F}_{\mathbb{L}_{\mathfrak{A}}}(X)$ and hence $\overline{\mathbb{L}}_{\mathfrak{A}} = \bigcup\{X \mid X \subseteq \mathcal{F}_{\mathbb{L}_{\mathfrak{A}}}(X)\}$.

The second way fixed points arise is as a justification of induction and coinduction proof principles.

To illustrate this, compare proving $\forall l \in \mathbb{L}_{\mathfrak{A}}.\, \theta_1(l) = \theta_2(l)$ by induction, where $\theta_1 : \mathbb{L}_{\mathfrak{A}} \to \mathcal{A}$ and $\theta_2 : \mathbb{L}_{\mathfrak{A}} \to \mathcal{A}$, with proving $\forall x \in \mathcal{C}.\, \phi_1(x) = \phi_2(x)$ by coinduction, where $\phi_1 : \mathcal{C} \to \overline{\mathbb{L}}_{\mathfrak{A}}$ and $\phi_2 : \mathcal{C} \to \overline{\mathbb{L}}_{\mathfrak{A}}$.

To prove $\forall l \in \mathbb{L}_{\mathfrak{A}}.\, \theta_1(l) = \theta_2(l)$ by induction, let $P = \{l \mid \theta_1(l) = \theta_2(l)\}$, then the proof of $\forall l \in \mathbb{L}_{\mathfrak{A}}.\, l \in P$ by induction on l consists of the base case $\mathsf{nil} \in P$ and the induction step $\forall l.\, l \in P \;\Rightarrow\; \forall \mathfrak{a} \in \mathfrak{A}.\, \mathsf{cons}(\mathfrak{a}, l) \in P$.

This induction argument can be seen as an application of least fixed points because the base and induction correspond to proving that $\mathcal{F}_{\mathbb{L}_{\mathfrak{A}}}(P) \subseteq P$, i.e. that P is a pre-fixed point of $\mathcal{F}_{\mathbb{L}_{\mathfrak{A}}}$, so as $\mathbb{L}_{\mathfrak{A}}$ is the least pre-fixed point of $\mathcal{F}_{\mathbb{L}_{\mathfrak{A}}}$ it follows that $\mathbb{L}_{\mathfrak{A}} \subseteq P$, hence $\forall l \in \mathbb{L}_{\mathfrak{A}}.\, l \in P$.

To prove $\forall x \in \mathcal{C}.\, \phi_1(x) = \phi_2(x)$ by coinduction, let $R = \{(\phi_1(x), \phi_2(x)) \mid x \in \mathcal{C}\}$, then the proof by coinduction consists of showing that R is a bisimulation, i.e. that for all x: either $\phi_1(x) = \mathsf{nil}$ and $\phi_2(x) = \mathsf{nil}$ or else $\phi_1(x) = \mathsf{cons}(\mathfrak{a}, \phi_1(x'))$ and $\phi_2(x) = \mathsf{cons}(\mathfrak{a}, \phi_2(x'))$ for some $\mathfrak{a} \in \mathfrak{A}$ and $x' \in \mathcal{C}$.

This coinductive argument can be seen as an application of greatest fixed points because the set of pairs $\mathsf{EQ}_{\overline{\mathbb{L}}_{\mathfrak{A}}} = \{(l, l) \mid l \in \overline{\mathbb{L}}_{\mathfrak{A}}\}$ is the greatest post-fixed point of $\mathcal{B}_{\overline{\mathbb{L}}_{\mathfrak{A}}}$ defined by:

$\mathcal{B}(R) = \{(\mathsf{nil}, \mathsf{nil})\} \cup \{(\mathsf{cons}(n, l_1), \mathsf{cons}(n, l_2)) \mid n \in \mathbb{N} \wedge (l_1, l_2) \in R\}$

$\mathsf{EQ}_{\overline{\mathbb{L}}_{\mathfrak{A}}}$ being a post-fixed point means $\mathsf{EQ}_{\overline{\mathbb{L}}_{\mathfrak{A}}} \subseteq \mathcal{B}_{\overline{\mathbb{L}}_{\mathfrak{A}}}(\mathsf{EQ}_{\overline{\mathbb{L}}_{\mathfrak{A}}})$ and being the greatest post-fixed point means that if $R \subseteq \mathcal{B}_{\overline{\mathbb{L}}_{\mathfrak{A}}}(R)$ then $R \subset \mathsf{EQ}_{\overline{\mathbb{L}}_{\mathfrak{A}}}$.

Another way of saying that $\mathsf{EQ}_{\overline{\mathbb{L}}_{\mathfrak{A}}}$ is the greatest post-fixed point of $\mathcal{B}_{\overline{\mathbb{L}}_{\mathfrak{A}}}$ is with the equation $\mathsf{EQ}_{\overline{\mathbb{L}}_{\mathfrak{A}}} = \bigcup\{R \mid R \subseteq \mathcal{B}_{\overline{\mathbb{L}}_{\mathfrak{A}}}(R)\}$.

R is a bisimulation if $R \subseteq \mathcal{B}_{\overline{\mathbb{L}}_{\mathfrak{A}}}(R)$. The principle of bisimulation coinduction is that if R is a bisimulation then $R \subseteq \mathsf{EQ}_{\overline{\mathbb{L}}_{\mathfrak{A}}}$, i.e. if R is a bisimulation then $\forall l_1\, l_2 \in \overline{\mathbb{L}}_{\mathfrak{A}}.\, (l_1, l_2) \in R \;\Rightarrow\; l_1 = l_2$.

Summary of fixed point proof rules for lists and colists

If $\phi_1, \phi_2 : \mathbb{L}_{\mathfrak{A}} \to \mathcal{A}$ then $\forall l \in \mathbb{L}_{\mathfrak{A}}.\, \phi_1(l) = \phi_2(l)$ is proved by induction using the rule:

$$\text{if } P = \{l \mid \phi_1(l) = \phi_2(l)\} \text{ and } \mathcal{F}_{\mathbb{L}_{\mathfrak{A}}}(P) \subseteq P \text{ then } \mathbb{L}_{\mathfrak{A}} \subseteq P$$

This expands to:

$$\begin{array}{l}\phi_1(\mathsf{nil}) = \phi_2(\mathsf{nil}) \wedge (\forall l \in \mathbb{L}_{\mathfrak{A}}.\, \phi_1(l) = \phi_2(l) \;\Rightarrow\; \forall \mathfrak{a} \in \mathfrak{A}.\, \phi_1(\mathsf{cons}(\mathfrak{a}, l)) = \phi_2(\mathsf{cons}(\mathfrak{a}, l))) \\ \;\Rightarrow\; \forall l \in \mathbb{L}_{\mathfrak{A}}.\, \phi_1(l) = \phi_2(l)\end{array}$$

If $\phi_1, \phi_2 : \mathcal{C} \to \overline{\mathbb{L}}_{\mathfrak{A}}$ then $\forall x \in \mathcal{C}.\, \phi_1(x) = \phi_2(x)$ is proved by coinduction using:

$$\text{if } R = \{(\phi_1(x), \phi_2(x)) \mid x \in \mathcal{C}\} \text{ and } R \subseteq \mathcal{B}_{\mathbb{L}_{\mathfrak{A}}}(R) \text{ then } R \subseteq \mathsf{EQ}_{\overline{\mathbb{L}}_{\mathfrak{A}}}$$

This expands to:

$$\begin{array}{l}(\forall x \in \mathcal{C}. \\ \quad (\phi_1(x) = \mathsf{nil} \;\wedge\; \phi_2(x) = \mathsf{nil}) \\ \quad \vee \\ \quad \exists \mathfrak{a} \in \mathfrak{A}.\, \exists x' \in \mathcal{C}.\, \phi_1(x) = \mathsf{cons}(\mathfrak{a}, \phi_1(x')) \;\wedge\; \phi_2(x) = \mathsf{cons}(\mathfrak{a}, \phi_2(x'))) \\ \;\Rightarrow\; \forall x \in \mathcal{C}.\, \phi_1(x) = \phi_2(x)\end{array}$$

Use in programming

Initial $\mathbb{F}$-algebras correspond to programming language datatypes. Compare the ingredients of the initial $\mathbb{F}_{\mathbb{N}}$-algebra of numbers:

$\mathbb{F}_{\mathbb{N}}(\mathbb{N}) = \mathbf{1} + \mathbb{N}, \quad 0 \in \mathbb{N}, \quad \mathsf{S} : \mathbb{N} \to \mathbb{N}$

with functional programming pseudocode for a datatype declaration on numbers:

`data` $\mathbb{N} = 0 \mid \mathsf{S}$ `of` $\mathbb{N}$

These contain essentially the same specifications. The `of` indicates that the thing before it is a constructor of data taking arguments of the type shown after it. If there is no `of`, then the element is a nullary constructor, i.e. a distinguished element of the datatype.

The ingredients of the initial $\mathbb{F}_{\mathbb{L}_{\mathfrak{A}}}$-algebra of lists of members of $\mathfrak{A}$ are:

$\mathbb{F}_{\mathbb{L}_{\mathfrak{A}}}(\mathbb{L}_{\mathfrak{A}}) = \mathbf{1} + (\mathfrak{A} \times \mathbb{L}_{\mathfrak{A}}), \quad \mathsf{nil} \in \mathbb{L}_{\mathfrak{A}}, \quad \mathsf{cons} : \mathfrak{A} \times \mathbb{L}_{\mathfrak{A}} \to \mathbb{L}_{\mathfrak{A}}$

and the pseudocode for a corresponding datatype declaration:

`data` $\mathbb{L}_{\mathfrak{A}} = \mathsf{nil} \mid \mathsf{cons}$ `of` $(\mathfrak{A} \times \mathbb{L}_{\mathfrak{A}})$

The values specified by `data` declarations consists of finite structures built from the distinguished elements by applying constructors, e.g. $\mathsf{S}(\mathsf{S}(\mathsf{S}(0)))$ or $[\mathfrak{a}_0, \mathfrak{a}_1, \mathfrak{a}_2]$, i.e. $\mathsf{cons}(\mathfrak{a}_0, \mathsf{cons}(\mathfrak{a}_1, \mathsf{cons}(a_2, \mathsf{nil})))$. Recursion is used to construct data, for example the list $[n, n{-}1, \ldots, 1]$ would be constructed by executing $\mathsf{CountDownFrom}(n)$, where:

$\mathsf{CountDownFrom}(n) = \texttt{if}\ n = 0\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}(n, \mathsf{CountDownFrom}(n{-}1))$

Codatatypes are less common, but compare the ingredients of the terminal $\mathbb{F}$-coalgebras for conumbers and colists.

$$\begin{array}{lll} \mathbb{F}_{\mathbb{N}}(\overline{\mathbb{N}}) \quad = \mathbf{1} + \overline{\mathbb{N}}, & \mathsf{is0} : \overline{\mathbb{N}} \nrightarrow \mathbf{1}, & \mathsf{P} : \overline{\mathbb{N}} \nrightarrow \overline{\mathbb{N}} \\ \mathbb{F}_{\mathbb{L}_{\mathfrak{A}}}(\overline{\mathbb{L}}_{\mathfrak{A}}) = \mathbf{1} + (\mathfrak{A} \times \overline{\mathbb{L}}_{\mathfrak{A}}), & \mathsf{null} : \overline{\mathbb{L}}_{\mathfrak{A}} \nrightarrow \mathbf{1}, & \mathsf{destcons} : \overline{\mathbb{L}}_{\mathfrak{A}} \nrightarrow \mathfrak{A} \times \overline{\mathbb{L}}_{\mathfrak{A}} \end{array}$$

with the made up pseudocode:

$$\begin{array}{l} \texttt{codata}\ \overline{\mathbb{N}} \ \ = \mathsf{null} \quad \& \quad \mathsf{P}\ \texttt{to}\ \overline{\mathbb{N}} \\ \texttt{codata}\ \overline{\mathbb{L}}_{\mathfrak{A}} = \mathsf{null} \quad \& \quad \mathsf{destcons}\ \texttt{to}\ (\mathfrak{A} \times \overline{\mathbb{L}}_{\mathfrak{A}}) \end{array}$$

The $\mathbb{F}$-coalgebra specifications also contain essentially the same material as the pseudocode. The `to` indicates that the thing before it is a destructor that decomposes data into components of the type shown after it. If there is no `to`, then the element is a nullary destructor, i.e. a test for a distinguished element of the datatype.

For lists, hd and tl would normally be specified, rather than destcons. The made up pseudocode corresponding to:

$$\mathbb{F}_{\mathbb{L}_{\mathfrak{A}}}(\overline{\mathbb{L}}_{\mathfrak{A}}) = \mathbf{1} + (\mathfrak{A} \times \overline{\mathbb{L}}_{\mathfrak{A}}), \quad \mathsf{null} : \overline{\mathbb{L}}_{\mathfrak{A}} \nrightarrow \mathbf{1}, \quad \mathsf{hd} : \overline{\mathbb{L}}_{\mathfrak{A}} \nrightarrow \mathfrak{A}, \quad \mathsf{tl} : \overline{\mathbb{L}}_{\mathfrak{A}} \nrightarrow \overline{\mathbb{L}}_{\mathfrak{A}}$$

would be:

$$\texttt{codata}\ \overline{\mathbb{L}}_{\mathfrak{A}} = \mathsf{null} \quad \& \quad \mathsf{hd}\ \texttt{to}\ \mathfrak{A} \quad \& \quad \mathsf{tl}\ \texttt{to}\ \overline{\mathbb{L}}_{\mathfrak{A}}$$

The values specified by `codata` declarations may not be finite, so can't necessarily be represented explicitly in finite computer memories. However, these values can be implicitly represented and accessed incrementally by destructors, i.e. by lazy evaluation.

One way to define codata is by corecursion, for example

$\mathsf{CountFrom}(n) \ = \ \mathsf{cons}(n, \mathsf{CountFrom}(n{+}1))$

defines $\mathsf{CountFrom}(n)$ to be the infinite list starting from n. i.e. $[n, n{+}1, \ \ldots\ ]$. This corecursion is the instance of:

$g(x) = \texttt{if}\ \mathit{test}(x) = *\ \texttt{then}\ \mathsf{nil}\ \texttt{else}\ \mathsf{cons}((\mathsf{id} \times g)(\mathit{dest}(x)))$

where $\mathfrak{A} = \mathbb{N}$, $\mathit{test}(n) = *$ is always false and $\mathit{dest}(n) = (n, n{+}1)$. It is is the unique morphism from the $\mathbb{F}_{\mathbb{N}}$-coalgebra $(\mathbb{N}, \emptyset, \lambda n.\,(n, n{+}1))$ to the final $\mathbb{F}_{\mathbb{N}}$-coalgebra $(\overline{\mathbb{L}}_{\mathbb{N}}, \mathsf{null}, \mathsf{destcons})$, where $\mathsf{Dom}(\emptyset)$ is the empty set, so $\emptyset(n) = *$ is never true.

Another way codata is specified is by giving equations for the destructors, for example $\mathsf{CountFrom}(n)$ could be specified by:

$\mathsf{hd}(\mathsf{CountFrom}(n)) = n\ ;\ \mathsf{tl}(\mathsf{CountFrom}(n)) = \mathsf{CountFrom}(n{+}1)$

This style can be used to define codata corresponding to automata as already suggested by the discussion of the function g in the example of corecursion for lists used above and repeated in Figure 2 below.

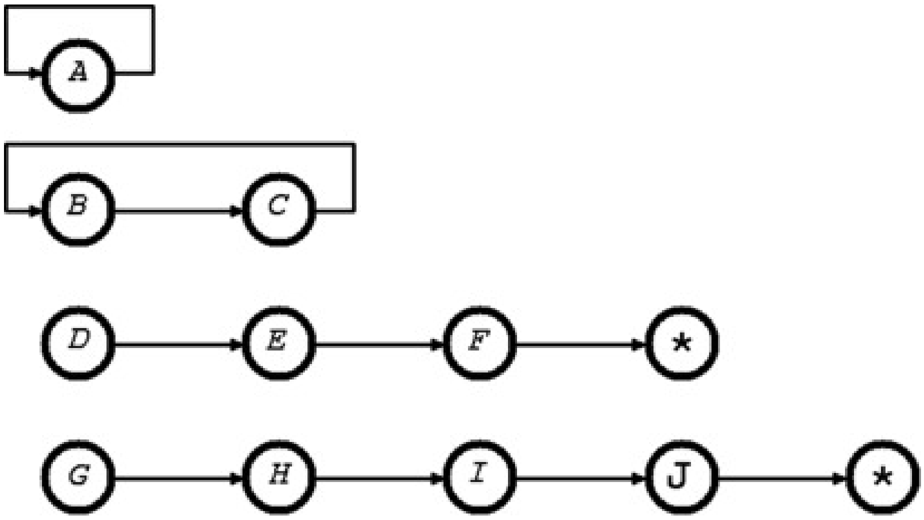

Figure 3:

The function g defined by this could be specified by:

$\mathsf{hd}(g(A))=A$; $\mathsf{hd}(g(B))=B$; $\mathsf{hd}(g(C))=C$; $\mathsf{hd}(g(D))=D$; $\mathsf{hd}(g(E))=E$;
$\mathsf{hd}(g(F))=F$; $\mathsf{hd}(g(G))=G$; $\mathsf{hd}(g(H))=H$; $\mathsf{hd}(g(I))=I$; $\mathsf{hd}(g(J))=J$;

$\mathsf{tl}(g(A))=g(A)$; $\mathsf{tl}(g(B))=g(C)$; $\mathsf{tl}(g(C))=g(B)$; $\mathsf{tl}(g(D))=g(E)$;
$\mathsf{tl}(g(E))=g(F)$; $\mathsf{tl}(g(G))=g(H)$; $\mathsf{tl}(g(H))=g(I)$; $\mathsf{tl}(g(I))=g(JA)$;

$\mathsf{null}(g(F))$; $\mathsf{null}(g(J))$

Unlike $\mathsf{CountFrom}$, which only creates infinite lists, the function g creates both infinite and finite lists: $g(A)$, $g(B)$ and $g(C)$ are infinite and $g(x)$ for $x \in \{D, E, F, G, H, I, J\}$ are finite, with $g(F)$ and $g(J)$ being nil.

There's an illuminating blog post that discusses Data vs Codata[25].

I've read that infinite data imported from external sources, e.g. from an analog-to-digital converter or a Twitter stream, can be considered to be codata. Presumably this view considers reading inputs as applying destructors, like hd and tl, so that coalgebra inspired programming methods can be used to process such imported data streams. Due to my near total ignorance, more will not be said on this now!

[25] http://www.tac-tics.net/blog/data-vs-codata

Concluding thoughts

I wrote this article as a way to learn about coinduction. Did I succeed? I think I did in that I now have a feeling – possibly delusional – of understanding the core ideas of coinduction and how it is dual to induction. I also now think I have a rough idea of the elementary parts of the general theory of algebras and coalgebras – at least the part that lives in set theory – and how this theory relates to recursion, induction, corecursion and coinduction. The most general formulations live in category theory – a territory in which I struggle to survive … but Google finds plenty of stuff, a random example being Worrell's PhD thesis[26] and there's an alluring motivational discussion in the Preface of *Introduction to Coalgebra*[27] by Bart Jacobs.

Most articles on coinduction aim to evangelise its use for applications. This is something I've pretty much ignored here. Particularly important applications are to reasoning about concurrent systems, indeed the Wikipedia article on coinduction[28] starts with the sentence "In computer science, coinduction is a technique for defining and proving properties of systems of concurrent interacting objects". In such applications the bisimulations that arise are often between labelled transition system[29]. As far as I am aware there are no significant applications of coinduction to reasoning about numbers and only a few to lists. The bulk of applications are to systems modelled with transition systems, so perhaps I should add something about these … but I'm burned out on coinduction and the tutorials cited at the beginning of this article are excellent, so I probably won't ever get around to adding anything on this.

First complete draft: February 03, 2017.

[26] http://www.cs.ox.ac.uk/people/james.worrell/thesis.ps
[27] https://goo.gl/BK0Pmr
[28] https://en.wikipedia.org/wiki/Coinduction
[29] https://en.wikipedia.org/wiki/Bisimulation