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

Mike Gordon
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>3</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>4</td>
</tr>
<tr>
<td>Data, codata, constructors and destructors</td>
<td>5</td>
</tr>
<tr>
<td>Recursion and corecursion</td>
<td>5</td>
</tr>
<tr>
<td>Fixed points, induction, coinduction and bisimulation</td>
<td>7</td>
</tr>
<tr>
<td>Overview of the rest of this article</td>
<td>8</td>
</tr>
<tr>
<td>Natural numbers, Peano’s axioms and Peano structures</td>
<td>9</td>
</tr>
<tr>
<td><strong>Algebras and coalgebras</strong></td>
<td>10</td>
</tr>
<tr>
<td>Numbers and conumbers</td>
<td>11</td>
</tr>
<tr>
<td>Example of corecursion for numbers</td>
<td>13</td>
</tr>
<tr>
<td>Lists and colists</td>
<td>14</td>
</tr>
<tr>
<td>Example of corecursion for lists</td>
<td>16</td>
</tr>
<tr>
<td><strong>Coinduction and bisimulation</strong></td>
<td>18</td>
</tr>
<tr>
<td>Bisimulation coinduction for conumbers</td>
<td>19</td>
</tr>
<tr>
<td>Example of bisimulation coinduction for numbers</td>
<td>19</td>
</tr>
<tr>
<td>Justification of bisimulation coinduction for numbers</td>
<td>20</td>
</tr>
<tr>
<td>Bisimulation coinduction for lists</td>
<td>21</td>
</tr>
<tr>
<td>Examples of bisimulation coinduction for lists</td>
<td>21</td>
</tr>
<tr>
<td>Justification of bisimulation coinduction for lists</td>
<td>25</td>
</tr>
<tr>
<td><strong>F-algebras and F-coalgebras</strong></td>
<td>26</td>
</tr>
<tr>
<td>Set theory notation</td>
<td>27</td>
</tr>
<tr>
<td>Numbers and lists</td>
<td>27</td>
</tr>
<tr>
<td>Morphisms</td>
<td>28</td>
</tr>
<tr>
<td>Initial and terminal algebras</td>
<td>30</td>
</tr>
<tr>
<td><strong>Least and greatest fixed points</strong></td>
<td>30</td>
</tr>
<tr>
<td>Fixed points, numbers and conumbers</td>
<td>31</td>
</tr>
<tr>
<td>Summary of fixed point proof rules for numbers and conumbers</td>
<td>32</td>
</tr>
<tr>
<td>Fixed points, lists and colists</td>
<td>32</td>
</tr>
<tr>
<td>Summary of fixed point proof rules for lists and colists</td>
<td>33</td>
</tr>
<tr>
<td><strong>Use in programming</strong></td>
<td>34</td>
</tr>
<tr>
<td><strong>Concluding thoughts</strong></td>
<td>36</td>
</tr>
</tbody>
</table>

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 generated from bespoke \LaTeX.
Preface

For years I’ve been vaguely aware of corecursion and coinduction as something dual to recursion and induction, and related to maximal fixed points and bisimulation – but I’ve never really grasped exactly what they are and what the “co” means.

I did some googling and found the link What the heck is coinduction and a couple of excellent tutorials:

- An introduction to (co)algebra and (co)induction by Bart Jacobs and Jan Rutten;

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

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.

The Wikipedia article on corecursion says:

... recursion works analytically, starting on data further from a base case and breaking it down into smaller data and repeating until one reaches a base case, corecursion works synthetically, starting from a base case and building it up, iteratively producing data further removed from a base case.

and the Wikipedia article on coinduction says:

Coinduction is the mathematical dual to structural induction. Coinductively defined types are known as codata and are typically infinite data structures, such as streams.

My goal in writing this article is to reduce my lack of understanding through the activity of creating a short and coherent explanation of the essence of corecursion and coinduction and how they relate to recursion and induction. The goal is not to show applications – these are covered in the tutorials mentioned above – the examples I give are not meant to testify to the utility of the concepts, just to illustrate them.
I mainly concentrate on corecursion and coinduction for numbers and lists. Numbers are the most well-known setting for recursion and induction, but corecursion and coinduction on numbers are pretty useless ... however, they do provide a simple setting for explaining the core ideas. I use lists as a second example as it generalises and extends the ideas first introduced for numbers, and provides a setting where corecusion and coinduction can be seen to be useful and are used in applications (though I don't discuss applications here). Lists are also a stepping stone to my regrettably superficial account of corecursion and coinduction in the more general setting of algebras and coalgebras.

The early drafts of this were rather formal and included proofs of everything. I even started to check some proofs using a computer proof assistant as I was struggling to understand the details and wanted to be confident I'd got them right. After a while I realised that writing the article 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. I decided to scale back on mathematical precision and largely give up on including formal proofs. A result of this is that I say things that I haven’t checked in detail, so there are bound to be errors, technical mistakes and embarrassing conceptual misunderstandings. If you happen to be reading this and spot any of these, then please let me know!

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 I happened to stumble 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

This section starts with three sub-sections giving a very condensed summary previewing of the core ideas in the rest of the article as they apply just to lists:

- 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.
A fourth sub-section itemising the topics covered in the remainder of the article concludes this summary.

The rest of the article 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 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 finite list \( l = [1, 1, 1, 1] \) is built by specifying \( l = \text{cons}(1, \text{cons}(1, \text{cons}(1, \text{cons}(1, \text{nil})))) \), where \( \text{cons} \) is a list constructor and \( \text{nil} \), the empty list, a nullary constructor.

- **Codata:** The infinite list \( l_{\infty} = [1, 1, 1, 1, \ldots] \) is defined by specifying \( \text{hd}(l_{\infty}) = 1 \) and \( \text{tl}(l_{\infty}) = l_{\infty} \), where \( \text{hd} \) and \( \text{tl} \) are destructors.

**Recursion and corecursion**

Recursion defines a function that maps values from a datatype by invoking itself on the components of the constructors used to build the datatype values. Corecursion defines a function to a codatatype by specifying the values of destructors on applications of the function. This is illustrated below using lists \( L \) and colists \( L_{\infty} \).

Let \( \mathbb{N} \) be the set of natural numbers, \( L \) be the set of finite lists of numbers and \( L_{\infty} \) the set of infinite lists of numbers. Let \( \mathbb{L} = L \cup L_{\infty} \) be the set of finite or infinite lists of numbers.

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

The list destructors are \( \text{null} \), \( \text{hd} \) and \( \text{tl} \): \( \text{null}(l) \) is true if and only \( l = \text{nil} \), \( \text{hd}(l) \) is the first element of \( l \) and \( \text{tl}(l) \) is the list resulting from removing its first element.

The equation \( l = \text{if} \ \text{null}(l) \ \text{then} \ \text{nil} \ \text{else} \ \text{cons}(\text{hd}(l), \text{tl}(l)) \) holds for every list \( l \).

**Example.** The function \( \text{Add1} \) adds 1 to each element of a list of numbers and is defined below both by recursion and corecursion. For the recursion one views \( \text{Add1} \) as having type \( \text{Add1} : L \rightarrow X \) and for corecursion having type \( \text{Add1} : X \rightarrow \mathbb{L} \). For the \( \text{Add1} \) example \( X = L \).
• Recursion:
\[ \text{Add1}(\text{nil}) = \text{nil} \text{ and } \text{Add1}(\text{cons}(n, l)) = \text{cons}(n+1, \text{Add1}(l)) \]

• Corecursion:
\[ \text{null}(\text{Add1}(l)) = (l=\text{nil}) \text{ and } \text{hd}(\text{Add1}(l)) = \text{hd}(l)+1 \text{ and } \text{tl}(\text{Add1}(l)) = \text{Add1}(\text{tl}(l)) \]

More generally, recursion defines functions from \( \mathbb{L} \), whereas corecursion defines functions to \( \overline{\mathbb{L}} \).

• If \( X \) is a set, \( x_0 \in X \) and \( \theta : \mathbb{N} \times X \rightarrow X \), then recursion legitimates the definition of \( f : \mathbb{L} \rightarrow X \) by:
\[ f(\text{nil}) = x_0 \text{ and } f(\text{cons}(n, l)) = \theta(n, f(l)) \]
which is equivalent to the single recursion equation:
\[ f(l) = \text{if } l = \text{nil} \text{ then } x_0 \text{ else } \theta(\text{hd}(l), f(\text{tl}(l))) \]

• If \( X \) is a set, \( P \subseteq X \) and \( \phi : \{ x \mid x \in X \land x \notin P \} \rightarrow \mathbb{N} \times X \), then corecursion legitimates the definition of \( g : X \rightarrow \overline{\mathbb{L}} \) by:
\[ \text{null}(g(x)) = (x \in P) \text{ and } \text{hd}(g(x)) = \pi_1(\phi(x)) \text{ and } \text{tl}(g(x)) = g(\pi_2(\phi(x))) \]
which is equivalent to the single corecursion equation:
\[ g(x) = \text{if } x \in P \text{ then } \text{nil} \text{ else cons}(\pi_1(\phi(x)), g(\pi_2(\phi(x)))) \]

where \( \pi_1, \pi_2 \) extract the first and second elements of a pair.

Returning to the \( \text{Add1} \) example equation:
\[ \text{Add1}(l) = \text{if } l = \text{nil} \text{ then } \text{nil} \text{ else cons}((\text{hd}(l)+1, \text{Add1}(\text{tl}(l)))) \]

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

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

The definition by recursion maps \( \mathbb{L} \) to \( \mathbb{L} \), but the definition by corecursion maps \( \overline{\mathbb{L}} \) to \( \overline{\mathbb{L}} \), which allows infinite lists to be arguments and results. This sub-section concludes with three examples of corecursion that will reappear later.

Example. The function \( \text{CountFrom} : \mathbb{N} \rightarrow \overline{\mathbb{E}} \) maps numbers to infinite lists, i.e. to the subset \( \mathbb{L}_\infty \) of \( \overline{\mathbb{L}} \).
\[ \text{CountFrom}(n) = \text{cons}(n, \text{CountFrom}(n+1)) \]

This corecursion equation defines \( \text{CountFrom}(n) \) to be the infinite list counting up from \( n \), i.e. \( \text{cons}(n, \text{cons}(n+1, \text{cons}(n+2, \ldots))) \). If \( X = \mathbb{N}, P \) is the empty set and \( \phi(n) = (n, n+1) \), then \( \text{CountFrom} = g \), where \( g \) is defined by corecursion by the single corecursion equation:
\[ g(x) = \text{if } x \in P \text{ then } \text{nil} \text{ else cons}(\pi_1(\phi(x)), g(\pi_2(\phi(x)))) \]
Thus if \( F \) is also the least pre-fixed point of \( \mathbb{L} \), i.e. to the subset \( \mathbb{L} \) of \( \mathbb{L} \) that is \( \subseteq \mathbb{L} \), the greatest fixed point is \( \mathbb{L} \). If \( X = \mathbb{N} \times \mathbb{N} \) and \( \phi(m,n) = (m,(m+1,n)) \), then \( \text{CountUp} = g \), where \( g \) is defined by corecursion by the single corecursion equation:

\[
g(x) = \text{if } x \in P \text{ then nil else cons}(\pi_1(\phi(x)), g(\pi_2(\phi(x))))
\]

**Example.** The function \( \text{CountUp} : \mathbb{N} \times \mathbb{N} \to \mathbb{L} \) maps pairs of numbers to both finite and infinite lists.

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

This corecursion equation defines \( \text{CountUp} \) to be the finite list consisting of successive numbers greater than or equal to \( m \) and less than \( n \), if \( m \leq n \), and the infinite list counting up from \( m \), if \( m > n \). If \( X = \mathbb{N} \times \mathbb{N} \), \( P = \{(m,n) \mid m = n\} \) and \( \phi(m,n) = (m,(m+1,n)) \), then \( \text{CountUp} = g \), where \( g \) is defined by corecursion by the single corecursion equation:

\[
g(x) = \text{if } x \in P \text{ then nil else cons}(\pi_1(\phi(x)), g(\pi_2(\phi(x))))
\]

**Example.** The function \( \text{CountUpTo} : \mathbb{N} \times \mathbb{N} \to \mathbb{L} \) maps pairs of numbers to finite lists, i.e. to the subset \( \mathbb{L} \) of \( \mathbb{L} \).

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

This corecursion equation defines \( \text{CountUpTo} \) to be the finite list consisting of successive numbers greater than or equal to \( m \) and less than \( n \), if \( m \leq n \), and nil if \( m > n \). If \( X = \mathbb{N} \times \mathbb{N} \), \( P = \{(m,n) \mid m \geq n\} \) and \( \phi(m,n) = (m,(m+1,n)) \), then \( \text{CountUpTo} = g \), where \( g \) is defined by corecursion by the single corecursion equation:

\[
g(x) = \text{if } x \in P \text{ then nil else cons}(\pi_1(\phi(x)), g(\pi_2(\phi(x))))
\]

**Example.** The function \( \text{CountUpTo} : \mathbb{N} \times \mathbb{N} \to \mathbb{L} \) maps pairs of numbers to finite lists, i.e. to the subset \( \mathbb{L} \) of \( \mathbb{L} \).

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

This corecursion equation defines \( \text{CountUpTo} \) to be the finite list consisting of successive numbers greater than or equal to \( m \) and less than \( n \), if \( m \leq n \), and nil if \( m > n \). If \( X = \mathbb{N} \times \mathbb{N} \), \( P = \{(m,n) \mid m \geq n\} \) and \( \phi(m,n) = (m,(m+1,n)) \), then \( \text{CountUpTo} = g \), where \( g \) is defined by corecursion by the single corecursion equation:

\[
g(x) = \text{if } x \in P \text{ then nil else cons}(\pi_1(\phi(x)), g(\pi_2(\phi(x))))
\]

**Example.** The function \( \text{CountUpTo} : \mathbb{N} \times \mathbb{N} \to \mathbb{L} \) maps pairs of numbers to finite lists, i.e. to the subset \( \mathbb{L} \) of \( \mathbb{L} \).

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

This corecursion equation defines \( \text{CountUpTo} \) to be the finite list consisting of successive numbers greater than or equal to \( m \) and less than \( n \), if \( m \leq n \), and nil if \( m > n \). If \( X = \mathbb{N} \times \mathbb{N} \), \( P = \{(m,n) \mid m \geq n\} \) and \( \phi(m,n) = (m,(m+1,n)) \), then \( \text{CountUpTo} = g \), where \( g \) is defined by corecursion by the single corecursion equation:

\[
g(x) = \text{if } x \in P \text{ then nil else cons}(\pi_1(\phi(x)), g(\pi_2(\phi(x))))
\]

**Fixed points, induction, coinduction and bisimulation**

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

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

\( \mathbb{L} \) is also the least pre-fixed point of \( \mathcal{F} \), that is the least \( L \) 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 \( F(L) \subseteq L \) is equivalent to \( \mathit{nil} \in L \) (the base case of the induction) and \( \forall n \in \mathbb{N}. \forall l \in L. \mathit{cons}(n, l) \in L \) (the induction step). \( \subseteq L \) is equivalent to \( \forall l \in \subseteq l \in L. \)

If \( R \subseteq \subseteq \times \subseteq \) is a binary relation on \( \subseteq \), then define \( B(R) \subseteq \subseteq \times \subseteq \) by:

\[
B(R) = \{ (\mathit{nil}, \mathit{nil}) \} \cup \{ (\mathit{cons}(n, l_1), \mathit{cons}(n, l_2)) \mid n \in \mathbb{N} \land (l_1, l_2) \in R \}
\]

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

The relation \( \{ (l, l) \mid l \in \subseteq \} \) is also the greatest post-fixed point of \( B \), that is the greatest \( R \) such that \( R \subseteq B(R) \). Thus if there exists an \( R \) such that \( R \subseteq B(R) \), then by greatestness \( R \subseteq \{ (l, l) \mid l \in \subseteq \} \).

The property \( R \subseteq B(R) \) means that if \( (l_1, l_2) \in R \) then either \( l_1 = \mathit{nil} \) and \( l_2 = \mathit{nil} \) or else \( l_1 = \mathit{cons}(n, l_1') \) and \( l_2 = \mathit{cons}(n, l_2') \) and \( (l_1', l_2') \in R \), for some \( n \in \mathbb{N} \) and \( l_1', l_2' \in \subseteq \). Such an \( R \) is called a \textit{bisimulation}. The proof principle that if \( R \) is a bisimulation, then \( \forall l_1, l_2 \in \subseteq. (l_1, l_2) \in R \Rightarrow l_1 = l_2 \) is an example of coinduction.

Recall the corecursively defined functions:

- \( \mathit{CountFrom}(n) = \mathit{cons}(n, \mathit{CountFrom}(n+1)) \)
- \( \mathit{CountUp}(m, n) = \text{if } m = n \text{ then } \mathit{nil} \text{ else } \mathit{cons}(m, \mathit{CountUp}(m+1, n)) \)
- \( \mathit{CountUpTo}(m, n) = \text{if } m \geq n \text{ then } \mathit{nil} \text{ else } \mathit{cons}(m, \mathit{CountUpTo}(m+1, n)) \)

Examples of bisimulations are \( R_1 \) and \( R_2 \), where:

- \( R_1 = \{ (\mathit{CountUp}(m, n), \mathit{CountFrom}(m)) \mid m > n \} \)
- \( R_2 = \{ (\mathit{CountUp}(m, n), \mathit{CountUpTo}(m, n)) \mid m \leq n \} \)

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 \( R_1 \):

\( \forall m, n \in \mathbb{N}. m > n \Rightarrow \mathit{CountUp}(m, n) = \mathit{CountFrom}(m) \).

By bisimulation coinduction applied to \( R_2 \):

\( \forall m, n \in \mathbb{N}. m \leq n \Rightarrow \mathit{CountUp}(m, n) = \mathit{CountUpTo}(m, n) \).

By combining these results of applying bisimulation coinduction to \( R_1 \) and \( R_2 \), the equation below can be deduced.

\( \mathit{CountUp}(m, n) = \text{if } m \leq n \text{ then } \mathit{CountUpTo}(m, n) \text{ else } \mathit{CountFrom}(m) \)

Both the terms “bisimulation” and “coinduction” originate from Robin Milner. For further historical details see Section 4.3 of Sangiorgi’s paper \textit{On the Origins of Bisimulation and Coinduction} and Milner and Tofte’s paper \textit{Co-induction in relational semantics}.

\textbf{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.

F-algebras and 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.

Natural numbers, Peano’s axioms and Peano structures

The Wikipedia article 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 \((\mathbb{N}, 0, S)\), where \(\mathbb{N}\) is a set, \(0 \in \mathbb{N}\) a constant and \(S : \mathbb{N} \to \mathbb{N}\) a one-argument function.

The five Peano axioms define the set \(\mathbb{N}\) of natural numbers:

1. \(0 \in \mathbb{N}\)
2. \(\forall n \in \mathbb{N}. \ S(n) \in \mathbb{N}\)
3. \(\forall n \in \mathbb{N}. \ S(n) \neq 0\)
4. \(\forall m \in \mathbb{N}. \forall n \in \mathbb{N}. \ S(m) = S(n) \Rightarrow m = n\)
5. ∀M. 0 ∈ M ∧ (∀n ∈ M. S(n) ∈ M) ⇒ N ⊆ M

The structure (N, 0, S) is an instance of a class of structures (A, z, s), where A is a set, z ∈ A and s : A → A. These structures seem to have several names, including discrete dynamical systems and Peano structures. The latter name is used here.

The existence of (N, 0, S) satisfying Peano’s axioms follows from the axioms of set theory, e.g. see this wikipedia article. Peano’s axioms entail the principle of recursive definition. This says that for any Peano structure (A, z, s) there is exactly one function f : N → A such that f(0) = z and ∀n ∈ N, f(S(n)) = s(f(n)). Showing this is straightforward, but not entirely trivial (e.g. see here for a discussion and here for a detailed proof).

The principle of recursive definition is equivalent to Peano’s axioms because (N, 0, S) satisfies the five Peano axioms if and only if for all Peano structures (A, z, s) there is exactly one function f : N → A such that: f(0) = z ∧ ∀n ∈ N, f(S(n)) = s(f(n)). (N, 0, S) an example of an initial algebra.

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.

It seems that algebras and coalgebras can be formulated in various ways. The simplest one I’ve found 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. Thus 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(a₁, . . . , aₙ) = a, then its dual destructor, d say, splits a into (a₁, . . . , aₙ), i.e. d(a) = (a₁, . . . , aₙ). In general, the dual of a non-nullary constructor is a partial function—it’s 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 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 $F$-algebras and terminal $F$-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 and this, which are random examples I found with Google.

### Numbers and conumbers

An example of an algebra is a Peano algebra, which is just what was previously called a Peano structure. A Peano algebra $(A, z, s)$ has a carrier $A$, only one distinguished element $z \in A$ and only one constructor function $s : A \to A$. 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 $(\mathbb{N}, 0, S)$ with the property that for any Peano structure $(A, z, s)$ there’s exactly one function $f : \mathbb{N} \to A$ such that $f(0) = z$ and $\forall n \in \mathbb{N}. f(S(n)) = s(f(n))$. The function $f$ can also be defined by a single equation $f(n) = \text{if } n=0 \text{ then } z \text{ else } s(f(n-1))$.

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

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 $\ast$, 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 $F$-algebras and $F$-coalgebras below. In the summary section $x \in d$ is an abbreviation for $x \in \text{Dom}(d)$, where $d$ is a nullary destructor.

The conumbers dual of the distinguished number 0 is the partial function $\text{id} : \{ 0 \} \to \{ \ast \}$, and so necessarily $\text{id}(0) = \ast$ and $\text{Dom}(\text{id}) = \{ 0 \}$.

If a coalgebra has more than one destructor, then all its destructors are partial functions. Here’s some useful notation. Writing $\theta : X \rightsquigarrow Y$ means $\theta$ is a partial function from set $X$ to set $Y$. The subset of $X$ where $\theta$ is defined –
i.e. the domain of $\theta$ is denoted by $\text{Dom}(\theta)$. If $\theta : X \to Y$ – i.e. $\theta$ is a total function – then $\text{Dom}(\theta) = X$. If $\theta : X \to Y$ 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 $\theta : X \to Y$ then $\theta : \text{Dom}(\theta) \to Y$.

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

The conatural numbers are the Peano coalgebra $(\mathbb{N},\text{isz0},\text{P})$ with the property that for any Peano coalgebra $(C,\text{isz},p)$ there is exactly one function $g : C \to \mathbb{N}$ such that for all $x \in C$:

- if $x \in \text{Dom}(\text{isz})$ then $g(x) \in \text{Dom}(\text{isz0})$ and $\text{isz0}(g(x)) = \text{isz}(x)$;
- if $x \in \text{Dom}(p)$ then $g(x) \in \text{Dom}(\text{P})$ and $\text{P}(g(x)) = g(p(x))$.

The coalgebra $(\mathbb{N},\text{isz0},\text{P})$ is an example of a terminal coalgebra.

It turns out that the unique existence of $g$ property in the preceding paragraph determines the Peano coalgebra $(\mathbb{N},\text{isz0},\text{P})$ to have carrier set $\mathbb{N} = \mathbb{N} \cup \{\infty\}$, the nullary destructor $\text{isz0} : \{0\} \to \{\ast\}$ satisfying $\text{isz0}(0) = \ast$ and the unary destructor $\text{P}$ to be the predecessor function $n \mapsto n-1$ extended by defining $\text{P}(\infty) = \infty$, thus $\text{P} : \{n \mid (n \in \mathbb{N} \land n > 0) \lor n = \infty\} \to \mathbb{N}$.

It’s useful to extend the addition operator to conumbers $\mathbb{N}$ by specifying that if either argument is $\infty$ then so is the result. For example, $\infty + 1 = \infty$. Note that $+$ extended to $\mathbb{N}$ is associative and commutative. With this extension, $m = \text{P}(n) \iff m+1 = n$ for all $m$ and $n$ in $\mathbb{N}$.

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

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

- constructors $0 \in \mathbb{N}$, $S : \mathbb{N} \to \mathbb{N}$ versus
  destructors $\text{isz0} : \mathbb{N} \to \{\ast\}$, $\text{P} : \mathbb{N} \to \mathbb{N}$;
- unique $f : \mathbb{N} \to A$ versus
  unique $g : C \to \mathbb{N}$;
- define $f$ on constructed values: $f(0) = z \land f(S(n)) = s(f(n))$ versus
  define destructors on values of $g$: $\text{isz0}(g(x)) = \text{isz}(x) \land \text{P}(g(x)) = g(p(x))$;
- recursion $f(n) = \text{if } n=0 \text{ then } z \text{ else } s(f(n-1))$ versus
  corecursion $g(x) = \text{if isz}(x) = \ast \text{ then } 0 \text{ else } g(p(x))+1$.
- simple recursion on natural numbers always terminates versus
  corecursion on conumbers sometimes doesn’t terminate (see example below).
Example of corecursion for numbers

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

\begin{align*}
C &= \{A, B, C, D, E, F, G, H, I, J\}, \\
isz &= [F \mapsto \ast, J \mapsto \ast], \\
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{align*}

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

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

Consider now the definition of $g : C \rightarrow \mathbb{N}$ specified for $x \in C$ by $g(x) = \text{if isz}(x) = \ast \text{ then } 0 \text{ else } g(p(x)) + 1$.

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

\begin{align*}
g(A) &= g(p(A)) + 1 = g(A) + 1
\end{align*}

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

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

\begin{align*}
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
\end{align*}

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

13
Thus evaluating $g(x)$ by rewriting with the equation for $g$ loops if $x \in \{A, B, C\}$.

On other arguments the rewriting terminates:

\[
\begin{align*}
g(F) &= 0 \\
g(E) &= g(p(E)) + 1 = g(F) + 1 = 0 + 1 = 1 \\
g(D) &= g(p(D)) + 1 = g(E) + 1 = 1 + 1 = 2 \\
g(J) &= 0 \\
g(I) &= g(p(I)) + 1 = g(J) + 1 = 0 + 1 = 1 \\
g(H) &= g(p(H)) + 1 = g(I) + 1 = 1 + 1 = 2 \\
g(G) &= g(p(G)) + 1 = g(H) + 1 = 2 + 1 = 3
\end{align*}
\]

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 $g : \mathcal{C} \rightarrow \mathbb{N}$ satisfying the equation $g(x) = \text{if } isz(x) = * \text{ then } 0 \text{ else } g(p(x)) + 1$ is described by:

$g(x) = \left\{ \begin{array}{ll}
n \exists x_0 \cdots x_n. x = x_0 \land isz(x_n) = * \land \forall i < n. p(x_i) = x_{i+1} \\
\infty \text{ otherwise}
\end{array} \right.$

The example just given illustrates this.

**Lists and colists**

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

Peano-style axioms for lists are:

1. $\text{nil} \in L_{\mathfrak{A}}$
2. $\forall a \in \mathfrak{A}, \forall l \in L_{\mathfrak{A}}. \text{cons}(a, l) \in L_{\mathfrak{A}}$
3. $\forall a \in \mathfrak{A}, \forall l \in L_{\mathfrak{A}}. \text{cons}(a, l) \neq \text{nil}$
4. $\forall a_1, a_2 \in \mathfrak{A}, \forall l_1, l_2 \in L_{\mathfrak{A}}. \text{cons}(a_1, l_1) = \text{cons}(a_2, l_2) \Rightarrow a_1 = a_2 \land l_1 = l_2$
5. $\forall M. \text{nil} \in M \land (\forall a \in \mathfrak{A}, \forall l \in M. \text{cons}(a, l) \in M) \Rightarrow L_{\mathfrak{A}} \subseteq M$

Axiom 5 entails that if $l \in L_{\mathfrak{A}}$ then $l = \text{nil}$ or $l = \text{cons}(a, l')$ for some $a \in \mathfrak{A}$ and $l' \in L_{\mathfrak{A}}$ (proof: specialise $M$ to $\{l \mid l = \text{nil} \lor \exists a \in \mathfrak{A}. \exists l'. l = \text{cons}(a, l')\}$ in Axiom 5). This and Axiom 4 shows that there are destructors $\text{hd} : L_{\mathfrak{A}} \rightarrow \mathfrak{A}$ and $\text{tl} : L_{\mathfrak{A}} \rightarrow L_{\mathfrak{A}}$, which satisfy $\forall l \in L_{\mathfrak{A}}. l = \text{nil} \lor l = \text{cons}(\text{hd}(l), \text{tl}(l))$.
These five Peano-style axioms for lists are equivalent to the single property that if $A$ is a set, $nl \in A$ and $cs : A \times A \rightarrow A$, then there is a unique function $f : L_A \rightarrow A$ such that $\forall l \in L_A. f(l) = \text{if } l = \text{nil} \text{ then } nl \text{ else } cs(\text{hd}(l), f(\text{tl}(l))).$

A structure $(A, nl, cs)$ where $nl \in A$ and $cs : A \times A \rightarrow A$ will be called an $A$-list algebra. The dual concept is an $A$-list coalgebra $(C, \text{test}, \text{dest})$ where $C$ is the carrier set and $\text{test} : C \rightarrow \{\star\}$ and $\text{dest} : C \rightarrow A \times C$ are destructors whose domains partition the carrier $C$.

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

- 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 $\pi_1 : X_1 \times X_2 \rightarrow X_1$ and $\pi_2 : X_1 \times X_2 \rightarrow X_2$ are called projections.
- If $\theta_1 : X_1 \rightarrow Y_1$ and $\theta_2 : X_2 \rightarrow Y_2$, then $\theta_1 \times \theta_2 : X_1 \times X_2 \rightarrow Y_1 \times Y_2$ 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 $\theta_1$ and $\theta_2$. 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 dual of the $A$-list algebra $(\mathbb{L}_A, \text{nil}, \text{cons})$ is the $A$-list coalgebra $(\mathbb{L}_A, \text{null}, \text{destcons})$ with the property that for any $A$-list coalgebra $(C, \text{test}, \text{dest})$, there is exactly one function $g : C \rightarrow \mathbb{L}_A$ such that for all $x \in C$:

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

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

It is useful to extend cons, hd and tl from $\mathbb{L}_A$ to all of $\mathbb{L}_A \cup A^N$. For $\sigma \in A^N$ define:
Consider now the definition of \( x \) in the diagram in Figure 1 below: an arrow from \( \text{Dom}(x) \) \( \mapsto \{A,B,C,D,E,F,G,H,I,J\} \) versus 

destructors \( \text{null} : \mathbb{L}_A \mapsto \{\ast\} \) \( \), \( \text{destcons} : \mathbb{L}_A \mapsto A \times \mathbb{L}_A \); 

- unique \( f : \mathbb{L}_A \mapsto A \) versus 
  unique \( g : C \mapsto \mathbb{L}_A \); 

- define \( f \) on constructors: \( f(\text{nil}) = \text{nil} \land f(\text{cons}(a,l)) = \text{cs}(a,f(l)) \) versus 
  define destructors on \( g \) : \( \text{null}(g(x)) = \text{test}(x) \land \text{destcons}(g(x)) = (\text{id} \times g)(\text{dest}(x)) \); 

- recursion \( f(l) = \text{if } l = \text{nil} \text{ then nil else cs}(\text{hd}(l), f(\text{tl}(l))) \) versus 
  corecursion \( g(x) = \text{if } \text{test}(x) = \ast \text{ then nil else cons}( (\text{id} \times g)(\text{dest}(x)) ) \). 

- 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 \( \mathbb{L}_A \) coalgebra.

\( \mathbb{A} = \{A,B,C,D,E,F,G,H,I,J\} \), 
\( \mathbb{C} = \{A,B,C,D,E,F,G,H,I,J\} \), 
\( \text{test} = [F \mapsto \ast, J \mapsto \ast] \), 
\( \text{dest} = [A \mapsto (A,A), B \mapsto (B,C), C \mapsto (C,B), D \mapsto (D,E), E \mapsto (E,F), 
G \mapsto (G,H), H \mapsto (H,I), I \mapsto (I,J)] \).

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

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

Consider now the definition of \( g : C \mapsto \mathbb{L}_A \) specified for \( x \in C \) by 
\( g(x) = \text{if } \text{test}(x) = \ast \text{ then nil else cons}( (\text{id} \times g)(\text{dest}(x)) ) \).
Figure 2:

The notation $[a_0, a_1, \ldots, a_n]$ abbreviates $\text{cons}(a_0, \text{cons}(a_1, \ldots, \text{cons}(a_n, \text{nil}) \cdots))$ and $[]$ is used as a synonym for nil.

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

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

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

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

$$g(B) = \text{cons}((\text{id} \times g)(\text{dest}(B)))$$
$$= \text{cons}((\text{id} \times g)(B, C))$$
$$= \text{cons}(B, g(C))$$
$$= \text{cons}(B, \text{cons}((\text{id} \times g)(\text{dest}(C))))$$
$$= \text{cons}(B, \text{cons}(C, g(B)))$$
$$= [B, C, g(B)]$$

$$g(C) = \text{cons}((\text{id} \times g)(\text{dest}(C)))$$
$$= \text{cons}((\text{id} \times g)(C, B))$$
$$= \text{cons}(C, g(B))$$
$$= \text{cons}(C, \text{cons}((\text{id} \times g)(\text{dest}(B))))$$
$$= \text{cons}(C, \text{cons}(B, g(C)))$$
$$= [C, B, g(C)]$$

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 \( \mathbb{2}^\mathbb{N} \).

On other arguments the rewriting results in a finite list, i.e. a member of \( \mathbb{1}_\alpha \).

The rewriting calculations below take bigger steps than the ones above (the expansion of \((\text{id} \times g)(\cdots)\) is omitted).

\[
\begin{align*}
g(F) &= \text{nil} = [] \\
g(E) &= \text{cons}(E, g(F)) = \text{cons}(E, \text{nil}) = [E] \\
g(D) &= \text{cons}(D, g(E)) = \text{cons}(D, \text{cons}(E, \text{nil})) = [D, E] \\
g(J) &= \text{nil} = [] \\
g(I) &= \text{cons}(I, g(J)) = \text{cons}(I, \text{nil}) = [I] \\
g(H) &= \text{cons}(H, g(I)) = \text{cons}(H, \text{cons}(I, \text{nil})) = [H, I] \\
g(G) &= \text{cons}(G, g(H)) = \text{cons}(G, \text{cons}(H, \text{cons}(I, \text{nil}))) = [G, H, I]
\end{align*}
\]

These rewriting calculations show that:

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

For an arbitrary \( \alpha \)-list coalgebra \((\mathcal{C}, \text{test}, \text{dest})\), the function \( g : \mathcal{C} \to \mathbb{1}_\alpha \) satisfying the equation \( g(x) = \text{if } \text{test}(x) = * \text{ then } \text{nil} \text{ else } \text{cons}((\text{id} \times g)(\text{dest}(x))) \) is described by:

\[
g(x) = \begin{cases} 
[a_0, \ldots, a_{n-1}] \exists x_0 \cdots x_n. \\
\quad x = x_0 \land \text{test}(x_n) = * \land \forall i < n. \text{dest}(x_i) = (a_i, x_{i+1}) \\
[a_0, a_1, a_2, \ldots] \exists \sigma : \mathbb{N} \to \mathcal{C}. x = \sigma(0) \land \forall i. \text{dest}(\sigma(i)) = (a_i, \sigma(i+1))
\end{cases}
\]

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 \( \mathbb{1}_\alpha \) 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* 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 here **bisimulation coinduction**, where a bisimulation is a relation \( R \) between members of the carrier of a coalgebra. Often this principle is just called coinduction. Bisimulation coinduction is derived from the corecursion equation defining the unique functions from arbitrary coalgebras to the conumber coalgebra \( \mathbb{N} \) or to the colist coalgebra \( \mathbb{L} \).

Bisimulation coinduction for conumbers

For the conatural numbers \( \mathbb{N} \), a bisimulation is a relation \( R \subseteq \mathbb{N} \times \mathbb{N} \) such that if \( (n_1, n_2) \in R \) then either \( n_1 = 0 \) and \( n_2 = 0 \) or else \( n_1 \neq 0 \) and \( n_2 \neq 0 \) and \((P(n_1), P(n_2)) \in R\).

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

**Example of bisimulation coinduction for numbers**  It’s hard to come up with examples for \( \mathbb{N} \) 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 more convincing example used later to illustrate coinduction on lists. In the example \( \text{even}(n) \) means that \( n \) is an even number (\( \text{even}(0) \) is considered true) and \( \text{odd}(n) \) that \( n \) is odd.

Consider the unique function \( g : \mathbb{N} \to \mathbb{N} \) determined by the Peano coalgebra \((\mathbb{N}, \text{isz}, p)\) where \( \text{Dom}(\text{isz}) = \{n \mid \text{even}(n)\} \) and \( \text{Dom}(p) = \{n \mid \text{odd}(n)\} \), and the destructors are defined by \( p(n) = * \iff \text{even}(n) \) and \( s(n) = n+2 \). This function \( g \) satisfies:

\[
g(x) = \text{if isz}(x) = * \text{ then 0 else } g(p(x))+1
\]

With the particular \( \text{isz} \) and \( p \) just specified, this is equivalent to:

\[
g(n) = \text{if even}(n) \text{ then 0 else } g(n+2)+1
\]

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

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 \( n_1 = g(n+2)+1 \) and \( n_2 = \infty \), so \( n_1 \neq 0 \) and \( n_2 \neq 0 \). Also, if \( n \) is not even then \( n+2 \) is not even, so \( P(n_1) = P(g(n)) = P((g(n)+2)+1) = g(n+2) \) and \( P(n_2) = P(h(n)) = P(\infty) = \infty = h(n+2) \), and so \((P(n_1), P(n_2)) \in R \).

Thus either \( n_1 = 0 \) and \( n_2 = 0 \) or \( n_1 \neq 0 \) and \( n_2 \neq 0 \) and \((P(n_1), P(n_2)) \in R \), 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 \( R \subseteq \mathbb{N} \times \mathbb{N} \) be a bisimulation. Consider the Peano coalgebra \((R, isz_R, p_R)\), where \( isz_R(n_1, n_2) = * \iff n_1 = 0 \land n_2 = 0 \) and \( p_R(n_1, n_2) = (P(n_1), P(n_2)) \). 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 \((\mathbb{N}, is0, P)\), there is a unique function \( g : R \to \mathbb{N} \) such that for all \((n_1, n_2) \in R\):

\[
g(n_1, n_2) = \text{if } isz_R(n_1, n_2) = * \text{ then } 0 \text{ else } g(p_R(n_1, n_2))+1
\]
i.e. for all \((n_1, n_2) \in R\):

\[
g(n_1, n_2) = \text{if } n_1 = 0 \land n_2 = 0 \text{ then } 0 \text{ else } g(P(n_1), P(n_2))+1
\]

Recall the projection functions: \( \pi_1(\sigma_1, \sigma_2) = \sigma_1 \) and \( \pi_2(\sigma_1, \sigma_2) = \sigma_2 \). It is easy to verify the equation for \( g \) is satisfied with both \( g = \pi_1 \) and \( g = \pi_2 \).

Taking \( g = \pi_1 \):

\[
\pi_1(n_1, n_2) = \text{if } n_1 = 0 \land n_2 = 0 \text{ then } 0 \text{ else } \pi_1(P(n_1), P(n_2))+1
\]
which simplifies to:

\[
n_1 = \text{if } n_1 = 0 \land n_2 = 0 \text{ then } 0 \text{ else } P(n_1)+1
\]
which holds for all \((n_1, n_2) \in R\), since if \((n_1, n_1) \in R\) and \( n_1 = 0 \) then as \( R \) is a bisimulation \( n_2 = 0 \).

Now take \( g = \pi_2 \):

\[
\pi_2(n_1, n_2) = \text{if } n_1 = 0 \land n_2 = 0 \text{ then } 0 \text{ else } \pi_2(P(n_1), P(n_2))+1
\]
which simplifies to:

\[
n_2 = \text{if } n_1 = 0 \land n_2 = 0 \text{ then } 0 \text{ else } P(n_2)+1
\]
which also holds for all \((n_1, n_2) \in R\).

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

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

\[
g(n) = \text{if } n = 0 \text{ then } 0 \text{ else } g(n+2)+1
\]
\[ g(x) = \text{if } \text{isz}(x) = * \text{ then } 0 \text{ else } g(p(x)) + 1 \]

To prove that the bisimulation coinduction principle entails the uniqueness of corecursively specified functions, suppose that:

\[
g_1(x) = \text{if } \text{isz}(x) = * \text{ then } 0 \text{ else } g_1(p(x)) + 1 \\
g_2(x) = \text{if } \text{isz}(x) = * \text{ then } 0 \text{ else } g_2(p(x)) + 1
\]

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

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

If \( \text{isz}(x) = * \), then \( n_1 = g_1(x) = 0 \) and \( n_2 = g_2(x) = 0 \).

If \( \text{isz}(x) \neq * \), then \( n_1 = g_1(x) = g_1(p(x)) + 1 \) and \( n_2 = g_2(x) = g_2(p(x)) + 1 \), so \( n_1 \neq 0 \) and \( n_2 \neq 0 \). Also, if \( \text{isz}(x) \neq * \), then \( P(n_1) = P(g_1(x)) = P(g_1(p(x)) + 1) = g_1(p(x)) \) and \( P(n_2) = P(g_2(x)) = P(g_2(p(x)) + 1) = g_2(p(x)) \), and so \( (P(n_1), P(n_2)) \in R \).

Thus either \( n_1 = 0 \) and \( n_2 = 0 \) or \( n_1 \neq 0 \) and \( n_2 \neq 0 \) and \( (P(n_1), P(n_2)) \in R \), so \( R \) is a bisimulation, hence by the bisimulation coinduction principle: \( \forall x \in C, g_1(x) = g_2(x) \).

**Bisimulation coinduction for lists**

A bisimulation on \( \Sigma_\mathcal{A} \) is a relation \( R \subseteq \Sigma_\mathcal{A} \times \Sigma_\mathcal{A} \) such that if \( (\sigma_1, \sigma_2) \in R \) then either \( \sigma_1 = \text{nil} \) and \( \sigma_2 = \text{nil} \) or else \( \sigma_1 \neq \text{nil} \) and \( \sigma_2 \neq \text{nil} \) and \( \text{hd}(\sigma_1) = \text{hd}(\sigma_2) \) and \( (\text{tl}(\sigma_1), \text{tl}(\sigma_2)) \in R \).

The bisimulation coinduction principle for \( \Sigma_\mathcal{A} \) states that if \( R \) is any bisimulation, then \( R \subseteq \{ (\sigma, \sigma) \mid \sigma \in \Sigma_\mathcal{A} \} \).

**Examples of bisimulation coinduction for lists**

Recall the corecursively defined functions \( \text{CountFrom} \), \( \text{CountUp} \) and \( \text{CountUpTo} \) from the summary at the beginning of this article.

\[
\begin{align*}
\text{CountFrom}(n) &= \text{cons}(n, \text{CountFrom}(n+1)) \\
\text{CountUp}(m, n) &= \text{if } m = n \text{ then } \text{nil} \text{ else } \text{cons}(m, \text{CountUp}(m+1, n)) \\
\text{CountUpTo}(m, n) &= \text{if } m \geq n \text{ then } \text{nil} \text{ else } \text{cons}(m, \text{CountUpTo}(m+1, n))
\end{align*}
\]

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

\[
\begin{align*}
R_1 &= \{ (\text{CountUp}(m, n), \text{CountFrom}(m)) \mid m > n \} \\
R_2 &= \{ (\text{CountUp}(m, n), \text{CountUpTo}(m, n)) \mid m \leq n \}
\end{align*}
\]

This assertion is now verified.

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

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

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

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

The remaining examples in this section are adapted from similar ones in *An introduction to (co)algebra and (co)induction* by Bart Jacobs and Jan Rutten, except here lists can be finite or infinite, but in Jacobs & Rutton they are only infinite.

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

$$\sigma : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7 \quad a_8 \quad a_9 \quad a_{10} \quad a_{11} \quad \cdots$$

$$g_{\text{even}}(\sigma) : \quad a_0 \quad a_2 \quad a_4 \quad a_6 \quad a_8 \quad a_{10} \quad \cdots$$

$$g_{\text{odd}}(\sigma) : \quad a_1 \quad a_3 \quad a_5 \quad a_7 \quad a_9 \quad a_{11} \quad \cdots$$

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

**Lemma 1.** $\forall \sigma \in \mathbb{L}_\alpha. g_{\text{odd}}(\sigma) = g_{\text{even}}(\text{tl}(\sigma))$

**Lemma 2.** $\forall \sigma \in \mathbb{L}_\alpha. \sigma \neq \text{nil} \Rightarrow \text{tl}(g_{\text{even}}(\sigma)) = g_{\text{odd}}(\text{tl}(\sigma))$

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

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

$$C_{\text{even}} = (\mathbb{L}_\alpha, \text{test}_{\text{even}}, \text{dest}_{\text{even}})$$

$$C_{\text{odd}} = (\mathbb{L}_\alpha, \text{test}_{\text{odd}}, \text{dest}_{\text{odd}})$$

where

$$\text{test}_{\text{even}}(\sigma) = \ast \Leftrightarrow \sigma = \text{nil}$$

$$\text{test}_{\text{odd}}(\sigma) = \ast \Leftrightarrow \sigma = \text{nil} \text{ or } \text{tl}(\sigma) = \text{nil}$$

and

$$\text{dest}_{\text{even}}(\sigma) = (\text{hd}(\sigma), \text{if } \text{tl}(\sigma) = \text{nil \ then nil else } \text{tl}(\text{tl}(\sigma)))$$

$$\text{dest}_{\text{odd}}(\sigma) = (\text{hd}(\text{tl}(\sigma)), \text{tl}(\text{tl}(\sigma)))$$

22
Recall the general form of the unique corecursively specified function $g : \mathcal{C} \rightarrow \Sigma_\mathcal{A}$ from the carrier of a coalgebra $(\mathcal{C}, test, dest)$ to the carrier of $(\Sigma_\mathcal{A}, null, destcons)$:

$$g(x) = \begin{cases} test(x) = \ast & \text{then nil} \\ \text{else cons}(\text{id} \times g)(\text{dest}(x)) \end{cases}$$

For the coalgebras $C_{even}$ and $C_{odd}$, this general schema instantiates, respectively, to:

$$g_{even}(\sigma) = \begin{cases} \text{if } \sigma = \text{nil} & \text{then nil} \\ \text{else cons}(\text{hd}(\sigma), g_{even}(\text{tl}(\sigma))) \end{cases}$$

$$g_{odd}(\sigma) = \begin{cases} \text{if } \sigma = \text{nil} \text{ or } \text{tl}(\sigma) = \text{nil} & \text{then nil} \\ \text{else cons}(\text{hd}(\text{tl}(\sigma)), g_{odd}(\text{tl}(\sigma))) \end{cases}$$

These equations look intuitively correct.

To prove Lemma 1: $\forall \sigma \in \Sigma_\mathcal{A}. g_{odd}(\sigma) = g_{even}(\text{tl}(\sigma))$, let $R = \{(g_{odd}(\sigma), g_{even}(\text{tl}(\sigma))) \mid \sigma \in \Sigma_\mathcal{A}\}$. It is shown below that $R$ is a bisimulation.

If $(\sigma_1, \sigma_2) \in R$, then $\sigma_1 = g_{odd}(\sigma)$ and $\sigma_2 = g_{even}(\text{tl}(\sigma))$ for some $\sigma \in \Sigma_\mathcal{A}$. Two cases need to be considered.

1. If $\sigma = \text{nil}$ or $\text{tl}(\sigma) = \text{nil}$ then $\sigma_1 = \sigma_2 = \text{nil}$ by the definitions of $g_{even}$ and $g_{odd}$.

2. If $\sigma \neq \text{nil}$ and $\text{tl}(\sigma) \neq \text{nil}$ then $\sigma_1 = g_{odd}(\sigma) = \text{cons}(\text{hd}(\text{tl}(\sigma)), g_{odd}(\text{tl}(\sigma)))$ and $\sigma_2 = g_{even}(\text{tl}(\sigma)) = \text{cons}(\text{hd}(\text{tl}(\sigma)), g_{even}(\text{tl}(\text{tl}(\sigma))))$ by the definitions of $g_{odd}$ and $g_{even}$, respectively. Hence $\text{hd}(\sigma_1) = \text{hd}(\sigma_2)$ and $\text{tl}(\sigma_1) = g_{odd}(\text{tl}(\sigma))$ and $\text{tl}(\sigma_2) = g_{even}(\text{tl}(\text{tl}(\sigma)))$. Thus $(\text{tl}(\sigma_1), \text{tl}(\sigma_2)) \in R$.

As $(\sigma_1, \sigma_2) \in R$ if either $\sigma_1 = \sigma_2 = \text{nil}$ or else $\sigma_1 \neq \text{nil}$ and $\sigma_2 \neq \text{nil}$ and $\text{hd}(\sigma_1) = \text{hd}(\sigma_2)$ and $(\text{tl}(\sigma_1), \text{tl}(\sigma_2)) \in R$, it follows that $R$ is a bisimulation.

To prove Lemma 2: $\sigma \neq \text{nil} \Rightarrow \text{tl}(g_{even}(\sigma)) = g_{odd}(\text{tl}(\sigma))$, let $R = \{(\text{tl}(g_{even}(\sigma)), g_{odd}(\text{tl}(\sigma))) \mid \sigma \in \Sigma_\mathcal{A} \land \sigma \neq \text{nil}\}$. It is shown below that $R$ is a bisimulation.

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

1. If $\text{tl}(\sigma) = \text{nil}$ then $\sigma_1 = \sigma_2 = \text{nil}$ by the definitions of $g_{even}$ and $g_{odd}$.

2. If $\text{tl}(\sigma) \neq \text{nil}$ and $\text{tl}(\text{tl}(\sigma)) = \text{nil}$ then $\sigma_1 = \text{tl}(g_{even}(\sigma)) = g_{even}(\text{tl}(\sigma)) = \text{nil}$ and $\sigma_2 = g_{odd}(\text{tl}(\sigma)) = \text{nil}$ by the definitions of $g_{odd}$ and $g_{even}$, respectively.

3. If $\text{tl}(\sigma) \neq \text{nil}$ and $\text{tl}(\sigma) \neq \text{nil}$ and $\text{tl}(\text{tl}(\sigma)) \neq \text{nil}$ then $\sigma_1 = \text{tl}(g_{even}(\sigma)) = g_{even}(\text{tl}(\sigma))) = \text{cons}(\text{hd}(\text{tl}(\sigma)), g_{even}((\text{if } - \text{then } - \text{else}))$ and $\sigma_2 = g_{odd}(\text{tl}(\sigma)) = \text{cons}(\text{hd}(\text{tl}(\sigma)), g_{odd}(\text{tl}(\text{tl}(\sigma))))$ by the definitions of $g_{odd}$ and $g_{even}$. Hence $\text{hd}(\sigma_1) = \text{hd}(\sigma_2)$ and $\text{tl}(\sigma_1) = \text{tl}(g_{even}(\text{tl}(\sigma)))$ and $\text{tl}(\sigma_2) = g_{odd}(\text{tl}(\text{tl}(\sigma)))$. Thus $(\text{tl}(\sigma_1), \text{tl}(\sigma_2)) \in R$.  

23
As \((σ_1, σ_2) ∈ R\) if either \(σ_1 = σ_2 = \text{nil}\) or else \(σ_1 \neq \text{nil}\) and \(σ_2 \neq \text{nil}\) and \(\text{hd}(σ_1) = \text{hd}(σ_2)\) and \((\text{tl}(σ_1), \text{tl}(σ_2)) ∈ R\), it follows that \(R\) is a bisimulation.

The function \(\text{merge} : \prod_α × \prod_α → \prod_α\) interleaves two lists. It will be proved by bisimulation coinduction that \(∀ σ ∈ \prod_α. \text{merge}(\text{even}(σ), \text{odd}(σ)) = σ\). Before proving this, the function \(\text{merge}\) needs to be defined. The natural recursion to achieve this is:

\[
\text{merge}(σ_1, σ_2) =
\begin{cases}
  \text{nil} & \text{if } σ_1 = \text{nil} \\
  \text{merge}(σ_1, \text{tl}(σ_2)) & \text{else if } σ_1 = \text{nil} \\
  \text{cons}(\text{hd}(σ_1), \text{merge}(σ_2, \text{tl}(σ_1))) & \text{else }
\end{cases}
\]

To make this fit the corecursion format:

\[
g(x) = \text{if } \text{test}(x) = * \text{ then nil else cons}(\text{id} × g)(\text{dest}(x))
\]

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

\[
\text{merge}(σ_1, σ_2) =
\begin{cases}
  \text{nil} & \text{if } σ_1 = \text{nil} \text{ and } σ_2 = \text{nil} \\
  \text{nil} & \text{else if } \text{tl}(σ_1) = \text{nil} \\
  \text{cons}(\text{hd}(σ_1), \text{merge}(σ_2, \text{tl}(σ_1))) & \text{else }
\end{cases}
\]

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

\[
g(x) = \text{if } \text{test}(x) = * \text{ then nil else cons}(\text{id} × g)(\text{dest}(x))
\]

if \(g = \text{merge}\) and \(\text{test} : \{\text{nil}\} × \{\text{nil}\} → \{*\}\) is defined by:

\[
\text{test}(σ_1, σ_2) = * \iff σ_1 = \text{nil} \text{ and } σ_2 = \text{nil}
\]

and \(\text{dest} : \{(σ_1, σ_2) \mid σ_1 \neq \text{nil} \text{ or } σ_2 \neq \text{nil}\} → \mathbb{A} × (\prod_α × \prod_α)\) by:

\[
\text{dest}(σ_1, σ_2) =
\begin{cases}
  \text{nil} & \text{if } σ_1 = \text{nil} \\
  \text{hd}(σ_2), (σ_1, \text{tl}(σ_2)) & \text{else if } σ_1 = \text{nil} \\
  \text{hd}(σ_1), (σ_2, \text{tl}(σ_1))) & \text{else }
\end{cases}
\]

The reformulated equation for \(\text{merge}\) is then the unique function from the carrier of the coalgebra \((\prod_α × \prod_α, \text{test}, \text{dest})\) to the carrier of the coalgebra \((\prod_α, \text{null}, \text{destcons})\).

The original recursion for \(\text{merge}\) can be derived from the reformulated equation using \(σ_2 = \text{cons}(\text{hd}(σ_2), \text{tl}(σ_2))\) and \(\text{merge}(\text{nil}, \text{tl}(σ_2)) = \text{tl}(σ_2)\), where the latter is
an instance of $\forall \sigma \in \underline{\alpha}, \text{merge}(\text{nil}, \sigma) = \sigma$, which is proved by bisimulation coinduction by showing that

$$R = \{ (\text{merge}(\text{nil}, \sigma), \sigma) \mid \sigma \in \underline{\alpha} \}$$

is a bisimulation. To show this let $(\sigma_1, \sigma_2) \in R$, then

$$\sigma_1 = \text{merge}(\text{nil}, \sigma) \text{ and } \sigma_2 = \sigma \text{ for some } \sigma \in \underline{\alpha}.$$ 

If $\sigma = \text{nil}$, then $\sigma_1 = \sigma_2 = \text{nil}$. If $\sigma \neq \text{nil}$ then

$$\sigma_1 = \text{cons}((\text{hd}(\sigma), \text{merge}(\text{nil}, \text{tl}(\sigma)))),$$

so $\text{hd}(\sigma_1) = \text{hd}(\sigma)$ and $\text{tl}(\sigma_1) = \text{merge}(\text{nil}, \text{tl}(\sigma))$. Thus either $\sigma_1 = \sigma_2 = \text{nil}$ or else $\sigma_1 \neq \text{nil}$ and $\sigma_2 \neq \text{nil}$ and $\text{hd}(\sigma_1) = \text{hd}(\sigma_2)$ and $(\text{tl}(\sigma_1), \text{tl}(\sigma_2)) \in R$, so $R$ is a bisimulation.

To prove $\text{merge}(\text{even}(\sigma), \text{odd}(\sigma)) = \sigma$ for arbitrary $\sigma \in \underline{\alpha}$, it is sufficient to show that

$$R = \{ (\text{merge}(\text{even}(\sigma), \text{odd}(\sigma)), \sigma) \mid \sigma \in \underline{\alpha} \}$$

is a bisimulation. The argument follows.

If $(\sigma_1, \sigma_2) \in R$, then

$$\sigma_1 = \text{merge}(\text{even}(\sigma), \text{odd}(\sigma)) \text{ and } \sigma_2 = \sigma \text{ for some } \sigma \in \underline{\alpha}.$$ 

If $\sigma = \text{nil}$ then $\sigma_1 = \text{merge}(\text{nil}, \text{nil}) = \text{nil}$ and $\sigma_2 = \sigma = \text{nil}$.

If $\sigma \neq \text{nil}$ then $\text{even}(\sigma) \neq \text{nil}$ and $\text{hd}(\sigma_1) = \text{hd}(\text{even}(\sigma)) = \text{hd}(\sigma) = \text{hd}(\sigma_2)$. Also,

\[
\text{tl}(\sigma_1) = \text{merge}(\text{odd}(\sigma), \text{tl}(\text{even}(\sigma))) = \text{merge}(\text{even}(\text{tl}(\sigma)), \text{odd}(\text{tl}(\sigma))) \text{.}
\]

As $\sigma \neq \text{nil}$, $\text{tl}(\text{even}(\sigma)) = \text{odd}(\sigma)$, so $\text{tl}(\sigma_1) = \text{merge}(\text{even}(\text{tl}(\sigma)), \text{odd}(\text{tl}(\sigma)))$. Clearly $\text{tl}(\sigma_2) = \text{tl}(\sigma)$, thus $\text{hd}(\sigma_1) = \text{hd}(\sigma_2)$ and $(\text{tl}(\sigma_1), \text{tl}(\sigma_2)) \in R$, so $R$ is a bisimulation.

**Justification of bisimulation coinduction for lists** A bisimulation on $\underline{\alpha}$ is a relation $R \subseteq \underline{\alpha} \times \underline{\alpha}$ such that $(\sigma_1, \sigma_2) \in R$ if either $\sigma_1 = \text{nil}$ and $\sigma_2 = \text{nil}$ or else $\sigma_1 \neq \text{nil}$ and $\sigma_2 \neq \text{nil}$ and $\text{hd}(\sigma_1) = \text{hd}(\sigma_2)$ and $(\text{tl}(\sigma_1), \text{tl}(\sigma_2)) \in R$.

To establish that the uniqueness of corecursively specified functions entails the principle of bisimulation coinduction, let $R \subseteq \underline{\alpha} \times \underline{\alpha}$ be a bisimulation. Define

the $\underline{\alpha}$-list coalgebra $(R, \text{test}_R, \text{dest}_R)$ by:

$$\text{test}_R(\sigma_1, \sigma_2) = * \iff \sigma_1 = \text{nil} \land \sigma_2 = \text{nil} \land \text{dest}_R(\sigma_1, \sigma_2) = (\text{hd}(\sigma_1), (\text{tl}(\sigma_1), \text{tl}(\sigma_2))).$$

Note that if $(\sigma_1, \sigma_2) \in R$ then $\text{hd}(\sigma_1) = \text{hd}(\sigma_2)$, so in the definition of $\text{dest}_R$ in the last sentence $\text{hd}(\sigma_1)$ could have been $\text{hd}(\sigma_2)$.

The definition of a bisimulation ensures that the domains of $\text{test}_R$ and $\text{dest}_R$ partition the coalgebra carrier set $R$. By the defining property of

$$(\underline{\alpha}, \text{null}, \text{destcons}),$$

there is a unique function $g : R \to \underline{\alpha}$ such that for all $(\sigma_1, \sigma_2) \in R$:

$$g(\sigma_1, \sigma_2) = \text{if } \text{test}_R(\sigma_1, \sigma_2) = * \text{ then nil else cons}((\text{id} \times g)(\text{dest}_R(\sigma_1, \sigma_2)))$$

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

$$g(\sigma_1, \sigma_2) = \text{if } \sigma_1 = \text{nil} \land \sigma_2 = \text{nil} \text{ then nil else cons}(\text{hd}(\sigma_1), g(\text{tl}(\sigma_1), \text{tl}(\sigma_2)))$$

It is easy to verify the equation for $g$ is satisfied with both $g = \pi_1$ and $g = \pi_2$. Taking $g = \pi_1$:

$$\pi_1(\sigma_1, \sigma_2) = \text{if } \sigma_1 = \text{nil} \land \sigma_2 = \text{nil} \text{ then nil else cons}(\text{hd}(\sigma_1), \pi_1(\text{tl}(\sigma_1), \text{tl}(\sigma_2)))$$

which simplifies to:

$$\sigma_1 = \text{if } \sigma_1 = \text{nil} \land \sigma_2 = \text{nil} \text{ then nil else cons}(\text{hd}(\sigma_1), \text{tl}(\sigma_1))$$
which holds for all $(\sigma_1, \sigma_2) \in R$. Now take $g = \pi_2$:

$$\pi_2(\sigma_1, \sigma_2) = \text{if } \sigma_1 = \text{nil} \land \sigma_2 = \text{nil} \text{ then nil else cons}(\text{hd}(\sigma_1), \pi_2(\text{tl}(\sigma_1), \text{tl}(\sigma_2)))$$

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

$$\sigma_2 = \text{if } \sigma_1 = \text{nil} \land \sigma_2 = \text{nil} \text{ then nil else cons}(\text{hd}(\sigma_2), \text{tl}(\sigma_2))$$

which also holds for all $(\sigma_1, \sigma_2) \in R$.

Since $g$ is unique, $\pi_1 : R \to \Sigma_\mathcal{X}$ and $\pi_2 : R \to \Sigma_\mathcal{Y}$ must be the same function, so if $(\sigma_1, \sigma_2) \in R$ then $\sigma_1 = \pi_1(\sigma_1, \sigma_2) = \pi_2(\sigma_1, \sigma_2) = \sigma_2$.

The argument just given shows that the bisimulation coinduction principle follows from the uniqueness of the function $g : \mathcal{X} \to \Sigma_\mathcal{Y}$ corecursively specified by:

$$g(x) = \text{if } \text{test}(x) = \ast \text{ then nil else cons}((\text{id} \times g)(\text{dest}(x)))$$

To prove that the bisimulation coinduction principle entails the uniqueness of corecursively specified functions, suppose that:

$$g_1(x) = \text{if } \text{test}(x) = \ast \text{ then nil else cons}((\text{id} \times g_1)(\text{dest}(x)))$$

$$g_2(x) = \text{if } \text{test}(x) = \ast \text{ then nil else cons}((\text{id} \times g_2)(\text{dest}(x)))$$

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

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

If $\text{test}(x) = \ast$, then $\sigma_1 = g_1(x) = \text{nil}$ and $\sigma_2 = g_2(x) = \text{nil}$.

If $\text{test}(x) \neq \ast$, then $\sigma_1 = g_1(x) = \text{cons}((\text{id} \times g_1)(\text{dest}(x))) = \text{cons}(\pi_1(\text{dest}(x)), g_1(\pi_2(\text{dest}(x))))$ and $\sigma_2 = g_2(x) = \text{cons}((\text{id} \times g_2)(\text{dest}(x))) = \text{cons}(\pi_1(\text{dest}(x)), g_2(\pi_2(\text{dest}(x))))$, so $\sigma_1 \neq \text{nil}$ and $\sigma_2 \neq \text{nil}$. Also, if $\text{hd}(\sigma_1) = \text{hd}(g_1(x)) = \pi_1(\text{dest}(x))$ and $\text{hd}(\sigma_2) = \text{hd}(g_2(x)) = \pi_1(\text{dest}(x))$, so $\text{hd}(\sigma_1) = \text{hd}(\sigma_2)$ and $\text{tl}(\sigma_1) = \text{tl}(g_1(x)) = \pi_1(\pi_2(\text{dest}(x)))$ and $\text{tl}(\sigma_2) = \text{tl}(g_2(x)) = \pi_2(\pi_2(\text{dest}(x)))$, hence $(\text{tl}(\sigma_1), \text{tl}(\sigma_2)) \in R$.

Thus either $\sigma_1 = \text{nil}$ and $\sigma_2 = \text{nil}$ or else $\sigma_1 \neq \text{nil}$ and $\sigma_2 \neq \text{nil}$ and $\text{hd}(\sigma_1) = \text{hd}(\sigma_2)$ and $(\text{tl}(\sigma_1), \text{tl}(\sigma_2)) \in R$, so $R$ is a bisimulation.

**F-algebras and F-coalgebras**

F-algebras and F-coalgebras are a uniform framework with numbers and lists being special cases. An introduction to (co)algebra and (co)induction by Bart 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.

26
Set theory notation

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

- The function \( \text{id}_X : X \rightarrow X \) is the identity: \( \forall x \in X. \text{id}_X(x) = x \).
- If \( \theta_1 : X \rightarrow Y \) and \( \theta_2 : Y \rightarrow Z \), then \( \theta_2 \circ \theta_1 : X \rightarrow Z \) is the function composition defined by \( \forall x \in X. (\theta_2 \circ \theta_1)(x) = \theta_2(\theta_1(x)) \). Note that \( \text{id}_Y \circ \theta_1 = \theta_1 \) and \( \theta_1 \circ \text{id}_X = \theta_1 \).
- 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 \( \theta_1 : X_1 \rightarrow Y_1 \) and \( \theta_2 : X_2 \rightarrow Y_2 \), then \( \theta_1 + \theta_2 : X_1 + X_2 \rightarrow Y_1 + Y_2 \) is defined by: \( (\theta_1 + \theta_2)(x) = \text{if } x \in X_1 \text{ then } \theta_1(x) \text{ else } \theta_2(x) \).
- 1 denotes the single-element set \( \{\ast\} \). It’s assumed that \( \ast \) isn’t a member of any of the carrier sets in the examples, so that the sums in the definitions of \( \mathbb{F}_N \) and \( \mathbb{F}_{\mathcal{A} \times \mathcal{A}} \) below are between disjoint sets.

Numbers and lists

The particular \( \mathbb{F} \)s for natural numbers and lists are \( \mathbb{F}_N \) and \( \mathbb{F}_{\mathcal{A} \times \mathcal{A}} \), defined by:

\[
\begin{align*}
\mathbb{F}_N(X) &= 1 + X \\
\mathbb{F}_{\mathcal{A} \times \mathcal{A}}(X) &= 1 + (\mathcal{A} \times X)
\end{align*}
\]

The \( \mathbb{F} \) lists that are only infinite is:

\[
\mathbb{F}_{\mathcal{A} \times \mathcal{A}}(X) = \mathcal{A} \times X
\]

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

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

The \( \mathbb{F}_N \)-algebra \( a : \mathbb{F}_N(\mathcal{A}) \rightarrow \mathcal{A} \) represents the Peano algebra \( (\mathcal{A}, z, s) \) where \( a \) is the function defined by: \( a(\ast) = z \) and \( a(x) = s(x) \) when \( x \in \mathcal{A} \).

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

Note how the nullary operators – the distinguished elements \( z \) and \( nl \) – are represented as function from 1 to the carrier.
An $\mathbb{F}$-colgebra is a pair $(C, c)$ where $c : C \to \mathbb{F}(C)$.

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

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

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

### 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 $\theta$ as illustrated by the following examples.

- If $X = Y = 1$ then necessarily $\theta = \text{id}_1$ and then $\mathbb{F}(\theta) = \mathbb{F}(\text{id}_1) = \text{id}_1$.
- If $\mathbb{F}(X) = 1 + X$ then $\mathbb{F}(\theta) : 1 + X \to 1 + Y$ and $\mathbb{F}(\theta) = \text{id}_1 + \theta$.
- If $\mathbb{F}(X) = 1 + (\mathbb{A} \times X)$ then $\mathbb{F}(\theta) : 1 + (\mathbb{A} \times X) \to 1 + (\mathbb{A} \times Y)$ and $\mathbb{F}(\theta) = \text{id}_1 + (\text{id}_{\mathbb{A}} \times \theta)$.

A function $f : A_1 \to A_2$ is a morphism from an $\mathbb{F}$-algebra $(A_1, a_1)$ to an $\mathbb{F}$-algebra $(A_2, a_2)$ 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 $\mathbb{F}_N$-algebra morphism $f : A_1 \to A_2$ from an $\mathbb{F}_N$-algebra $(A_1, a_1)$ corresponding to the Peano algebra $(A_1, z_1, s_1)$ to an $\mathbb{F}_N$-algebra $(A_2, a_2)$ corresponding to the Peano algebra $(A_2, z_2, s_2)$.

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

Expanding the definition of $\mathbb{F}_N(f)$ converts this equation to $f \circ a_1 = a_2 \circ (\text{id}_1 + f)$, which means $\forall x \in (1 + A_1). f(a_1(x)) = a_2((\text{id}_1 + f)(x))$.

Now, if $x \in (1 + A_1)$ then either $x \in 1$ or $x \in A_1$, so there are two cases to consider.

i. If $x \in 1$ then $x = *$ and $a_1(x) = z_1$ so $f(a_1(x)) = f(z_1)$. If $x = *$ then $(\text{id}_1 + f)(x) = \text{id}_1(x) = x = *$, so $a_2((\text{id}_1 + f)(x)) = a_2(\ast) = z_2$. Thus the equation $f(a_1(x)) = a_2((\text{id}_1 + f)(x))$ reduces to $f(z_1) = z_2$.

28
ii. If \( x \in A_1 \) then \( a_1(x) = s_1(x) \), so \( f(a_1(x)) = f(s_1(x)) \). If \( x \in A_1 \) then 
\[
(id_1 + f)(x) = f(x), \text{ so } a_2(id_1 + f)(x) = a_2(f(x)) = s_2(f(x)).
\]
Thus the equation \( f(a_1(x)) = a_2(id_1 + f)(x) \) reduces to \( f(s_1(x)) = s_2(f(x)) \).

Thus a morphism from the \( F_N \)-algebra corresponding to the natural numbers \( (N, 0, S) \) to an \( F_N \)-algebra corresponding to \((A, z, s)\) is a function \( f : N \to A \) such that \( f(0) = z \) and \( \forall n \in N. f(S(n)) = s(f(n)) \), i.e.: 
\[
f(n) = \text{if } n = 0 \text{ then } z \text{ else } s(f(n-1)).
\]
This is the recursive equation characterising the natural numbers.

A function \( g : C_1 \to C_2 \) is a morphism from an \( F \)-coalgebra \((C_1, c_1)\) to an \( F \)-coalgebra \((C_2, c_2)\) if and only if \( c_2 \circ g = F(g) \circ c_1 \).

To see what morphisms between coalgebras are, consider \( F_N \)-coalgebras. Recall: 
\[
F_N(X) = 1 + X.
\]
Suppose \( g : C_1 \to C_2 \) is a morphism from an \( F_N \)-coalgebra \( c_1 : C_1 \to F_N(C_1) \) corresponding to a Peano coalgebra \((C_1, isz_1, p_1)\) to an \( F_N \)-coalgebra \( c_2 : C_2 \to F_N(C_2) \) corresponding to a Peano coalgebra \((C_2, isz_2, p_2)\). There are two cases:

- if \( x \in \text{Dom}(isz_1) \) then \( c_1(x) = *; \) if \( x \in \text{Dom}(p_1) \) then \( c_1(x) \in C_1 \) and \( c_1(x) = p_1(x) \);
- if \( x \in \text{Dom}(isz_2) \) then \( c_2(x) = *; \) if \( x \in \text{Dom}(p_2) \) then \( c_2(x) \in C_2 \) and \( c_2(x) = p_2(x) \).

If \( g : C_1 \to C_2 \) is a morphism then \( c_2(g(x)) = F_N(g)(c_1(x)) \) holds. As \( F_N(g) = id_1 + g \), the right hand side of the morphism equation can be simplified corresponding to the two cases above:

- if \( x \in \text{Dom}(isz_1) \) then \( F_N(g)(c_1(x)) = (id_1 + g)(*) = id X(*) = *; \)
- if \( x \in \text{Dom}(p_1) \) then \( F_N(g)(c_1(x)) = F_N(g)(p_1(x)) = g(p_1(x)); \)

so for these two cases the morphism equation simplifies to:

- if \( x \in \text{Dom}(isz_1) \) then \( c_2(g(x)) = *; \)
- if \( x \in \text{Dom}(p_1) \) 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 \( g(x) \notin \text{Dom}(isz_2) \) and hence \( g(x) \in \text{Dom}(p_2) \) and then \( c_2(g(x)) = p_2(g(x)) \), so the two cases of the morphism equation further simplify to:

- if \( x \in \text{Dom}(isz_1) \) then \( isz_2(g(x)) = *; \)
- if \( x \in \text{Dom}(p_1) \) then \( p_2(g(x)) = g(p_1(x)). \)

29
Specialising the above discussion: a morphism from an \( F_N \)-coalgebra corresponding to \((C, isz, p)\) to the \( F_N \)-coalgebra corresponding to the conatural numbers \((\mathbb{N}, is0, P)\) is a function \( g : A \rightarrow \mathbb{N} \) such that

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

Thus a morphism to the \( F_N \)-colgebra corresponding to the conatural numbers \((\mathbb{N}, is0, P)\) from \( F_N \)-colgebra corresponding to \((C, isz, p)\) is a function \( g : C \rightarrow \mathbb{N} \) such that: \( g(x) = \text{if } isz(x) = * \text{ then } 0 \text{ else } g(p(x))+1 \). This is the corecursion equation characterising the conatural numbers.

**Initial and terminal algebras**

An *initial* \( F \)-algebra is one for which there is a unique morphism from it to any other \( F \)-algebra. The natural numbers are characterised as being the unique initial \( F_N \)-algebra and the finite lists of members of \( A \) are characterised as being the unique initial \( F_{LA} \)-algebra.

A *terminal* \( F \)-coalgebra is one for which there is a unique morphism to it from any other \( F \)-coalgebra. Terminal coalgebras are sometimes called *final* coalgebras. The conatural numbers are characterised as being the unique terminal \( F_N \)-coalgebra and the finite and infinite lists of elements of \( A \) are characterised as being the unique terminal \( F_{LA} \)-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* by A.D. Gordon) whilst others are based on initial and terminal algebras (e.g. *An introduction to (co)algebra and (co)induction* 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 \( F \)-algebras and terminal \( F \)-coalgebras for a wide class of \( F \)'s. Very roughly, the idea is that a least fixed point of \( F \) yields an initial \( F \)-algebra and a greatest fixed point yields a terminal \( F \)-coalgebra. 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*).

30
Fixed points, numbers and conumbers

A hint of the idea can be glimpsed by looking at numbers. If the mapping $\mathcal{F}_N$ from subsets of $\mathbb{N}$ to subsets of $\mathbb{N}$ is defined by:

$$\mathcal{F}_N(X) = \{0\} \cup \{S(x) \mid x \in X\}$$

then $\mathbb{N}$ and $\overline{\mathbb{N}}$ are both fixed points of $\mathcal{F}_N$, but $\mathbb{N}$ is the least fixed point and is the greatest fixed point.

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

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

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

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 $\forall n. n \in P \Rightarrow S(n) \in P$.

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

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

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

$$B_{\mathbb{N}}(R) = \{(n_1, n_2) \mid n_1 = 0 \land n_2 = 0 \lor n_1 \neq 0 \land n_2 \neq 0 \land (P(n_1), P(n_2)) \in R\}$$

$$B_{\overline{\mathbb{N}}}(R) = \{(0, 0)\} \cup \{(S(n_1), S(n_2)) \mid (n_1, n_2) \in R\}$$

Clearly $EQ \subseteq B_{\overline{\mathbb{N}}}(EQ)$ hence: $EQ = \bigcup\{R \mid R \subseteq B_{\overline{\mathbb{N}}}(R)\}$. $R$ being a bisimulation is just $R \subseteq B_{\mathbb{N}}(R)$, so $R$ is less than the greatest such post-fixed point, so $R \subseteq EQ$, hence if $(n_1, n_2) \in R$ then $(n_1, n_2) \in EQ$ and so $n_1 = n_2$. 
Summary of fixed point proof rules for numbers and conumbers

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

\[
\text{if } P = \{ n \mid \theta_1(n) = \theta_2(n) \} \text{ and } F_{\mathbb{N}}(P) \subseteq P \text{ then } \mathbb{N} \subseteq P
\]

This expands to:

\[
\theta_1(0) = \theta_2(0) \land (\forall n \in \mathbb{N}. \theta_1(n) = \theta_2(n) \Rightarrow \theta_1(S(n)) = \theta_2(S(n))) \Rightarrow \forall n \in \mathbb{N}. \theta_1(n) = \theta_2(n)
\]

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

\[
\text{if } R = \{ (\phi_1(x), \phi_2(x)) \mid x \in C \} \text{ and } R \subseteq B_{\mathbb{N}}(R) \text{ then } R \subseteq EQ
\]

This expands to:

\[
(\forall x \in C. \phi_1(x) = 0 \land \phi_2(x) = 0) \lor (\phi_1(x) \neq 0 \land \phi_2(x) \neq 0 \land (\exists x'. P(\phi_1(x)) = \phi_1(x') \land P(\phi_2(x)) = \phi_2(x')))) \Rightarrow \forall x \in C. \phi_1(x) = \phi_2(x)
\]

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 \( F_{L_A} \) from subsets of \( L_A \) to subsets of \( L_A \) is defined by:

\[
F_{L_A}(X) = \{ \text{nil} \} \cup \{ \text{cons}(a, x) \mid (a \in A \land x \in X) \}
\]

then \( L_A \) and \( L_A \) are both fixed points of \( F_{L_A} \), but \( L_A \) is the least fixed point and \( L_A \) is the greatest fixed point.

\( L_A \) is also the least pre-fixed point of \( F_{L_A} \), that is the least \( X \) such that \( F_{L_A}(X) \subseteq X \) and hence \( L_A = \bigcap \{ X \mid F_{L_A}(X) \subseteq X \} \).

Dually \( L_A \) is the greatest post-fixed point of \( F_{L_A} \), that is the greatest \( X \) such that \( X \subseteq F_{L_A}(X) \) and hence \( L_A = \bigcup \{ X \mid X \subseteq F_{L_A}(X) \} \).

The second way fixed points arise is as a justification of induction and coinduction proof principles.
To illustrate this, compare proving $\forall l \in \mathbb{L}_\alpha, \theta_1(l) = \theta_2(l)$ by induction, where $\theta_1 : \mathbb{L}_\alpha \to A$ and $\theta_2 : \mathbb{L}_\alpha \to A$, with proving $\forall x \in C, \phi_1(x) = \phi_2(x)$ by coinduction, where $\phi_1 : C \to \mathbb{L}_\alpha$ and $\phi_2 : C \to \mathbb{L}_\alpha$.

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

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

To prove $\forall x \in C, \phi_1(x) = \phi_2(x)$ by coinduction, let $R = \{(\phi_1(x), \phi_2(x)) \mid x \in C\}$, then the proof by coinduction consists of showing that $R$ is a bisimulation, i.e. that for all $x$: either $\phi_1(x) = \text{nil}$ and $\phi_2(x) = \text{nil}$ or else $\phi_1(x) \neq \text{nil}$ and $\phi_2(x) \neq \text{nil}$ and $\text{hd}(\phi_1(x)) = \text{hd}(\phi_2(x))$ and $(\text{tl}(\phi_1(x)), \text{tl}(\phi_2(x))) \in R$, i.e. $\exists x' \in C. \text{tl}(\phi_1(x)) = \phi_1(x') \land \text{tl}(\phi_2(x)) = \phi_2(x')$.

This coinductive argument can be seen as an application of greatest fixed points because the set of pairs $\text{EQ} = \{(l, l) \mid l \in \mathbb{L}_\alpha\}$ is the greatest post-fixed point of $B_{\mathbb{L}_\alpha} : \mathcal{P}(\mathbb{L}_\alpha \times \mathbb{L}_\alpha) \to \mathcal{P}(\mathbb{L}_\alpha \times \mathbb{L}_\alpha)$ defined by:

$B_{\mathbb{L}_\alpha}(R) = \{(l_1, l_2) \mid l_1 = \text{nil} \land l_2 = \text{nil} \lor l_1 \neq \text{nil} \land l_2 \neq \text{nil} \land \text{hd}(l_1) = \text{hd}(l_2) \land (\text{tl}(l_1), \text{tl}(l_2)) \in R\}$

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

Clearly $\text{EQ} \subseteq B_{\mathbb{L}_\alpha}(\text{EQ})$ hence: $\text{EQ} = \bigcup\{R \mid R \subseteq B_{\mathbb{L}_\alpha}(R)\}$. $R$ being a bisimulation is just $R \subseteq B_{\mathbb{L}_\alpha}(R)$, so $R$ is less than the greatest such post-fixed point, so $R \subseteq \text{EQ}$, hence if $(l_1, l_2) \in R$ then $(l_1, l_2) \in \text{EQ}$ and so $l_1 = l_2$.

---

**Summary of fixed point proof rules for lists and colists**

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

if $P = \{l \mid \phi_1(l) = \phi_2(l)\}$ and $F_{\mathbb{L}_\alpha}(P) \subseteq P$ then $\mathbb{L}_\alpha \subseteq P$

This expands to:

$\phi_1(\text{nil}) = \phi_2(\text{nil}) \land (\forall l \in \mathbb{L}_\alpha, \phi_1(l) = \phi_2(l) \Rightarrow \forall a \in A. \phi_1(\text{cons}(a, l)) = \phi_2(\text{cons}(a, l)))$

$\Rightarrow \forall l \in \mathbb{L}_\alpha, \phi_1(l) = \phi_2(l) - 33$
This expands to:

\[(\forall x \in C. \\
(\phi_1(x) = \text{nil} \land \phi_2(x) = \text{nil}) \\
\lor \\
(\phi_1(x) \neq \text{nil} \land \phi_2(x) \neq \text{nil} \land \text{hd}(\phi_1(x)) = \text{hd}(\phi_2(x)) \land \\
\exists x'. \text{tl}(\phi_1(x)) = \phi_1(x') \land \text{tl}(\phi_2(x)) = \phi_2(x'))) \\
\Rightarrow \forall x \in C. \phi_1(x) = \phi_2(x)\]

Use in programming

Initial \(F\)-algebras correspond to programming language datatypes. Compare the ingredients of the initial \(F_N\)-algebra of numbers:

\[F_N(N) = 1 + N, \quad 0 \in N, \quad S : N \to N\]

with functional programming pseudocode for a datatype declaration on numbers:

\[\text{data } N = 0 \mid S \text{ of } N\]

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

The ingredients of the initial \(F_{L_A}\)-algebra of lists of members of \(A\) are:

\[F_{L_A}(L_A) = 1 + (A \times L_A), \quad \text{nil} \in L_A, \quad \text{cons} : A \times L_A \to L_A\]

and the pseudocode for a corresponding datatype declaration:

\[\text{data } L_A = \text{nil} \mid \text{cons of } (A \times L_A)\]

The values specified by \text{data} declarations consists of finite structures built from the distinguished elements by applying constructors, e.g. \(S(S(S(0)))\) or \([a_0, a_1, a_2]\), i.e. \(\text{cons}(a_0, \text{cons}(a_1, \text{cons}(a_2, \text{nil})))\). Recursion is used to construct data, for example the list \([n, n-1, \ldots, 1]\) would be constructed by executing \text{downfrom}(n), where: \text{downfrom}(n) = if \(n = 0\) then \text{nil} else \text{cons}(n, \text{downfrom}(n-1)).

Codatatypes are less common, but compare the ingredients of the terminal \(F\)-coalgebras for conumbers and colists.

\[F_N(\mathbb{N}) = 1 + \mathbb{N}, \quad \text{is}0 : \mathbb{N} \to 1, \quad P : \mathbb{N} \to \mathbb{N}\]

\[F_{L_A}(L_A) = 1 + (A \times L_A), \quad \text{null} : L_A \to 1, \quad \text{destcons} : L_A \to A \times L_A\]

with the made up pseudocode:

\[\text{codata } \mathbb{N} = \text{null} \& P \text{ to } \mathbb{N}\]

\[\text{codata } L_A = \text{null} \& \text{destcons to } (A \times L_A)\]
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, \( \text{hd} \) and \( \text{tl} \) would normally be specified, rather than \( \text{destcons} \). The made up pseudocode corresponding to:

\[
\begin{align*}
F_{L^A}(\overline{g}) = 1 + (A \times \overline{g}), & \quad \text{null} : \overline{g} \to 1, \quad \text{hd} : \overline{g} \to A, \quad \text{tl} : \overline{g} \to \overline{g}\\
\end{align*}
\]

would be:

\[
\text{codata } \overline{A} = \text{null } & \& \text{ hd to } A & \& \text{ tl to } \overline{A}
\]

The values specified by \text{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

\[
\text{CountFrom}(n) = \text{cons}(n, \text{CountFrom}(n+1))
\]

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

\[
g(x) = \begin{cases} 
\text{nil} & \text{if } \text{test}(x) = * \\
\text{cons}(\text{id } g)(\text{dest}(x)) & \text{else}
\end{cases}
\]

where \( A = \mathbb{N} \), \( \text{test}(n) = * \) is always false and \( \text{dest}(n) = (n, n+1) \). 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 \((\overline{\mathbb{N}}, \text{null, destcons})\), where \( \text{Dom}(\emptyset) \) is the empty set, so \( \emptyset(n) = * \) is never true.

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

\[
\text{hd}(\text{CountFrom}(n)) = n ; \text{tl}(\text{CountFrom}(n)) = \text{CountFrom}(n+1)
\]

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.

The function \( g \) defined by this could be specified by:

\[
\begin{align*}
\text{hd}(g(A)) &= A; \quad \text{hd}(g(B)) = B; \quad \text{hd}(g(C)) = C; \quad \text{hd}(g(D)) = D; \quad \text{hd}(g(E)) = E; \\
\text{hd}(g(F)) &= F; \quad \text{hd}(g(G)) = G; \quad \text{hd}(g(H)) = H; \quad \text{hd}(g(J)) = J; \\
\text{tl}(g(A)) &= g(A); \quad \text{tl}(g(B)) = g(C); \quad \text{tl}(g(C)) = g(B); \quad \text{tl}(g(D)) = g(E); \\
\text{tl}(g(E)) &= g(F); \quad \text{tl}(g(G)) = g(H); \quad \text{tl}(g(H)) = g(I); \quad \text{tl}(g(I)) = g(JA); \\
\text{null}(g(F)); \quad \text{null}(g(J))
\end{align*}
\]

Unlike \( \text{CountFrom} \), 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.

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!

**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, which is territory in which I struggle to survive … but Google finds plenty of stuff, a random example being Worrell’s PhD thesis.

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 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. 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.