Coinduction, Corecursion, Copatterns

Anton Setzer

Swansea University, Swansea UK
(Joint work with Andreas Abel, Brigitte Pientka, David Thibodeau)

Swansea, 21 February 2013

Algebraic Data Types

In most functional programming languages we have the notion of an algebraic data type, e.g.

$$
\begin{array}{l}
\mathrm{data}\ \mathbb{N} : \mathrm{Set}\ \ \mathrm{where} \\
\quad \begin{array}{lcl}
\mathsf{0} & : & \mathbb{N} \\
\mathrm{S} & : & \mathbb{N} \to \mathbb{N}
\end{array}
\end{array}
$$

$$
\begin{array}{l}
\mathrm{data}\ \mathrm{NatList} : \mathrm{Set}\ \ \mathrm{where} \\
\quad \begin{array}{lcl}
\mathrm{nil} & : & \mathrm{NatList} \\
\mathrm{cons} & : & \mathbb{N} \to \mathrm{NatList} \to \mathrm{NatList}
\end{array}
\end{array}
$$

Algebraic Data Types as F-Algebras

$$\begin{array}{l} \text{data } \mathbb{N} : \text{Set where} \\ \quad \begin{array}{lcl} 0 & : & \mathbb{N} \\ \mathrm{S} & : & \mathbb{N} \to \mathbb{N} \end{array} \end{array}$$

can be rewritten as

$$\begin{array}{l} \text{data } \mathbb{N} : \text{Set where} \\ \quad \begin{array}{lcl} \text{intro} & : & (1 + \mathbb{N}) \to \mathbb{N} \end{array} \end{array}$$

or with $\mathrm{F}(X) := 1 + X$

$$\begin{array}{l} \text{data } \mathbb{N} : \text{Set where} \\ \quad \begin{array}{lcl} \text{intro} & : & \mathrm{F}(\mathbb{N}) \to \mathbb{N} \end{array} \end{array}$$

So $0 = \text{intro inl}$ and $\mathrm{S}\ n = \text{intro}\ (\text{inr}\ n)$.

Algebraic Data Types as F-Algebras

$$\begin{array}{l}\mathrm{data\ NatList : Set\ where} \\ \quad \mathrm{nil} \quad : \quad \mathrm{NatList} \\ \quad \mathrm{cons} \quad : \quad \mathbb{N} \to \mathrm{NatList} \to \mathrm{NatList}\end{array}$$

can we written as

$$\begin{array}{l}\mathrm{data\ NatList : Set\ where} \\ \quad \mathrm{nil} \quad : \quad 1 \to \mathrm{NatList} \\ \quad \mathrm{cons} \quad : \quad (\mathbb{N} \times \mathrm{NatList}) \to \mathrm{NatList}\end{array}$$

and with $\mathrm{F}(X) := 1 + (\mathbb{N} \times X)$ becomes

$$\begin{array}{l}\mathrm{data\ NatList : Set\ where} \\ \quad \mathrm{intro} \quad : \quad \mathrm{F}(\mathrm{NatList}) \to \mathrm{NatList}\end{array}$$

Initial F-Algebras

Initial F-Algebras F^* are minimal F-Algebras:

$$
\begin{tikzcd}
\mathrm{F}(\mathrm{F}^*) \arrow[r, "\mathrm{intro}"] \arrow[d, "\mathrm{F}(g)"'] & \mathrm{F}^* \arrow[d, "\exists!\, g"] \\
\mathrm{F}(A) \arrow[r, "f"] & A
\end{tikzcd}
$$

Iteration

Existence of g corresponds to iteration (example $\mathbb{N}$):

$$\begin{tikzcd}
1 + \mathbb{N} \arrow[r, "\mathrm{intro}"] \arrow[d, "1 + g"'] & \mathbb{N} \arrow[d, "\exists g"] \\
1 + A \arrow[r, "f"] & A
\end{tikzcd}$$

$$\begin{array}{lclcl}
g\ 0 & = & g\ (\mathrm{intro\ inl}) & = & f\ \mathrm{inl} \\
g\ (\mathrm{S}\ n) & = & g\ (\mathrm{intro}\ (\mathrm{inr}\ n)) & = & f\ (\mathrm{inr}\ (g\ n))
\end{array}$$

$$\begin{array}{lcl} g\ 0 & = & f\ \mathrm{inl} \\ g\ (\mathrm{S}\ n) & = & f\ (\mathrm{inr}\ (g\ n)) \end{array}$$

So with $a_0 := f\ \mathrm{inl} : A$ and $f_0 := f \circ \mathrm{inr} : A \to A$

$$\begin{array}{lcl} g\ 0 & = & a_0 \\ g\ (\mathrm{S}\ n) & = & f_0\ (g\ n) \end{array}$$

and therefore

$$g\ n = f_0^n\ a_0$$

On the other hand for every

$$a_0 : A \qquad f_0 : A \to A$$

we can define f and therefore g s.t. this equation holds.
So initial F-algebra means just **unique iteration**.

Recursion

The principle of recursion can be derived using uniqueness (I learned this from Thorsten Altenkirch):
Assume

$$\begin{array}{lcl} a_0 & : & A \\ f_0 & : & \mathbb{N} \to A \to A \end{array}$$

We derive $g : \mathbb{N} \to A$ s.t.

$$\begin{array}{lcl} g\ 0 & = & a_0 \\ g\ (\mathrm{S}\ n) & = & f_0\ n\ (g\ n) \end{array}$$

This allows to define efficiently the inverse of S:

$$\begin{array}{lcl} \mathrm{pred} & : & \mathbb{N} \to \mathbb{N} \\ \mathrm{pred}\ 0 & = & 0 \\ \mathrm{pred}\ (\mathrm{S}\ n) & = & n \end{array}$$

Recursion

We have

$$\begin{array}{lll} a_0 & : & A \\ f_0 & : & \mathbb{N} \to A \to A \end{array}$$

We need to have an F-algebra, we take as carrier

$$\mathbb{N} \times A$$

Define

$$\begin{array}{lll} f : (1 + (\mathbb{N} \times A)) \to (\mathbb{N} \times A) & & \\ f\ \mathrm{inl} & = & \langle 0, a_0 \rangle \\ f\ (\mathrm{inr}\ \langle n, a \rangle) & = & \langle \mathrm{S}\ n, f_0\ n\ a \rangle \end{array}$$

$$
\begin{aligned}
f\ \mathrm{inl} &= \langle 0, a\rangle \\
f\ (\mathrm{inr}\ \langle n, a\rangle) &= \langle n+1, f_0\ n\ a\rangle
\end{aligned}
$$

$$
\begin{tikzcd}
1+\mathbb{N} \arrow[r, "\mathrm{intro}"] \arrow[d, "1+g"'] & \mathbb{N} \arrow[d, "g"] \\
1+(\mathbb{N}\times A) \arrow[r, "f"] \arrow[d, "1+\pi_0"'] & \mathbb{N}\times A \arrow[d, "\pi_0"] \\
1+\mathbb{N} \arrow[r, "\mathrm{intro}"] & \mathbb{N}
\end{tikzcd}
$$

Both $\pi_0 \circ g$ and id make the outermost diagram commute.
By uniqueness follows $\pi_0 \circ g = \mathrm{id}$,
therefore $g\ n = \langle n, g_0\ n\rangle$ for some $g_0 : \mathbb{N} \to A$.

$$
\begin{aligned}
f\ \mathrm{inl} &= \langle 0, a \rangle \\
f\ (\mathrm{inr}\ \langle n, a \rangle) &= \langle n+1, f_0\ n\ a \rangle
\end{aligned}
$$

$$g\ n = \langle n, g_0\ n \rangle$$

$$
\begin{tikzcd}
1+\mathbb{N} \arrow[r, "\mathrm{intro}"] \arrow[d, "1+g"'] & \mathbb{N} \arrow[d, "g"] \\
1+(\mathbb{N}\times A) \arrow[r, "f"] & \mathbb{N}\times A
\end{tikzcd}
$$

Therefore

$$
\begin{aligned}
g_0\ 0 &= \pi_1(g\ (\mathrm{intro}\ \mathrm{inl})) &&= \pi_1(f\ \mathrm{inl}) &&= a_0 \\
g_0\ (\mathrm{S}\ n) &= \pi_1(g\ (\mathrm{intro}\ (\mathrm{inr}\ n))) &&= \pi_1(f\ (\mathrm{inr}\ \langle n, g_0\ n \rangle)) &&= f_0\ n\ (g_0\ n)
\end{aligned}
$$

Induction

Induction can be regarded as dependent elimination:
Assume

$$\begin{array}{lcl} A & : & \mathbb{N} \rightarrow \mathrm{Set} \\ a_0 & : & A\ 0 \\ f_0 & : & (n : \mathbb{N}) \rightarrow A\ n \rightarrow A\ (\mathrm{S}\ n) \end{array}$$

We derive $g : (n : \mathbb{N}) \rightarrow A\ n$ s.t.

$$\begin{array}{lcl} g\ 0 & = & a_0 \\ g\ (\mathrm{S}\ n) & = & f_0\ n\ (g\ n) \end{array}$$

Can be derived in the same way as recursion.

Initial Algebras in Functional Programming

Coalgebras and Copatterns

Codata Types

Conclusion

Coalgebras

Final coalgebras F^∞ are obtained by reversing the arrows in the diagram for F-algebras:

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "\exists! g"'] & \mathrm{F}(A) \arrow[d, "\mathrm{F}(g)"] \\ \mathrm{F}^\infty \arrow[r, "\mathrm{case}"] & \mathrm{F}(\mathrm{F}^\infty) \end{tikzcd}$$

Coalgebras

Consider Streams $= \mathrm{F}^\infty$ where $\mathrm{F}(X) = \mathbb{N} \times X$:

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "\exists! g"'] & \mathbb{N} \times A \arrow[d, "\mathrm{id} \times g"] \\ \mathrm{Stream} \arrow[r, "\mathrm{case}"] & \mathbb{N} \times \mathrm{Stream} \end{tikzcd}$$

Let

$$\mathrm{case}\ s \quad = \quad \langle \mathrm{head}\ s, \mathrm{tail}\ s \rangle$$

and

$$f\ \ a = \langle f_0\ \ a, f_1\ \ a \rangle$$

Guarded Recursion

$$\begin{tikzcd}
A \arrow[rr, "\langle f_0, f_1 \rangle"] \arrow[d, "\exists! g"'] & & \mathbb{N} \times A \arrow[d, "\mathrm{id} \times g"] \\
\mathrm{Stream} \arrow[rr, "\langle \mathrm{head}, \mathrm{tail} \rangle"] & & \mathbb{N} \times \mathrm{Stream}
\end{tikzcd}$$

Resulting equations:

$$\begin{array}{lcl}
\mathrm{head}\ (g\ a) & = & f_0\ a \\
\mathrm{tail}\ \ (g\ a) & = & g\ (f_1\ a)
\end{array}$$

Example of Guarded Recursion

$$
\begin{array}{lll}
\mathrm{head}\ (g\ a) & = & f_0\ a \\
\mathrm{tail}\ \ (g\ a) & = & g\ (f_1\ a)
\end{array}
$$

describes a schema of guarded recursion (or better coiteration)
As an example, with $A = \mathbb{N}$, $f_0\ n = n$, $f_1\ n = n + 1$ we obtain:

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathrm{inc} : \mathbb{N} \to \mathrm{Stream}} \\
\mathrm{head}\ (\mathrm{inc}\ n) & = & n \\
\mathrm{tail}\ \ (\mathrm{inc}\ n) & = & \mathrm{inc}\ (n + 1)
\end{array}
$$

Corecursion

In coiteration we need to make in tail always a recursive call:

$$\mathrm{tail}\ (g\ a) = g\ (f_1\ a)$$

Corecursion allows for tail to escape into a previously defined stream.
Assume

$$\begin{array}{lcl} A & : & \mathrm{Set} \\ f_0 & : & A \to \mathbb{N} \\ f_1 & : & A \to (\mathrm{Stream} + A) \end{array}$$

we get $g : A \to \mathrm{Stream}$ s.t.

$$\begin{array}{lllll} \mathrm{head}\ (g\ a) & = & f_0\ a & & \\ \mathrm{tail}\ \ (g\ a) & = & s & \text{if} & f_1\ a = \mathrm{inl}\ s \\ \mathrm{tail}\ \ (g\ a) & = & g\ a' & \text{if} & f_1\ a = \mathrm{inr}\ a' \end{array}$$

Iteration and Recursion

(I learned this symmetry from Peter Hancock)

Iteration: For $a_0 : A$, $f_0 : A \to A$ we get

$$\begin{array}{lcl} f : \mathbb{N} \to A & & \\ f\ 0 & = & a_0 \\ f\ (\mathrm{S}\ n) & = & f_0\ (f\ n) \end{array}$$

Recursion: For $a_0 : A$, $f_0 : (\mathbb{N} \times A) \to A$ we get

$$\begin{array}{lcl} f : \mathbb{N} \to A & & \\ f\ 0 & = & a_0 \\ f\ (\mathrm{S}\ n) & = & f_0\ \langle n, f\ n \rangle \end{array}$$

Coiteration and Corecursion

Iteration: For $f_0 : A \to \mathbb{N}$, $f_1 : A \to A$ we get

$$\begin{array}{lcl} f : A \to \mathrm{Stream} & & \\ \mathrm{head}\,(f\ a) & = & f_0\ a \\ \mathrm{tail}\,(f\ a) & = & f\ (f_1\ a) \end{array}$$

Corecursion: For $f_0 : A \to \mathbb{N}$, $f_1 : A \to (\mathrm{Stream} + A)$ we get

$$\begin{array}{lclll} f : A \to \mathrm{Stream} & & & & \\ \mathrm{head}\,(f\ a) & = & f_0\ a & & \\ \mathrm{tail}\,(f\ a) & = & s & \text{if} & f_1\ a = \mathrm{inl}\ s \\ \mathrm{tail}\,(f\ a) & = & f\ a' & \text{if} & f_1\ a = \mathrm{inr}\ a' \end{array}$$

Recursion, Corecursion

Recursion allows to define the inverse of the constructor S

$$
\begin{array}{lcl}
\mathrm{pred} : \mathbb{N} \to \mathbb{N} & & \\
\mathrm{pred}\ 0 & = & 0 \\
\mathrm{pred}\ (\mathrm{S}\ n) & = & n
\end{array}
$$

Corecursion allows to define the inverse of the destrutors head, tail:

$$
\begin{array}{lcl}
\mathrm{cons} : \mathbb{N} \to \mathrm{Stream} \to \mathrm{Stream} & & \\
\mathrm{head}\ (\mathrm{cons}\ n\ s) & = & n \\
\mathrm{tail}\ (\mathrm{cons}\ n\ s) & = & s
\end{array}
$$

Nested Corecursion

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\mathrm{stutter} : \mathbb{N} \to \mathrm{Stream}} \\
\mathrm{head}\ \ \ \ \ \ \ (\mathrm{stutter}\ n) & = & n \\
\mathrm{head}\ (\mathrm{tail}\ (\mathrm{stutter}\ n)) & = & n \\
\mathrm{tail}\ \ \ (\mathrm{tail}\ (\mathrm{stutter}\ n)) & = & \mathrm{stutter}\ (n+1)
\end{array}
$$

Even more general schemata can be defined.

Weakly Final Coalgebra

- Equality for final coalgebras is undecidable:
  Two streams

$$\begin{array}{rclllllll} s & = & (a_0 & , & a_1 & , & a_2 & , & \ldots \\ t & = & (b_0 & , & b_1 & , & b_2 & , & \ldots \end{array}$$

  are equal iff $a_i = b_i$ for all i.
- Even the weak assumption

$$\forall s.\exists n, s'.s = \mathrm{cons}\ n\ s'$$

  results in an undecidable equality.
- Weakly final coalgebras obtained by omitting uniqueness of g in diagram for coalgebras.
- However, one can extend schema of coiteration as above, and still preserve decidability of equality.
  - Those schemata are usually not derivable in weakly final coalgebras.

Patterns and Copatterns

- We can define now functions by patterns and copatterns.
- Example define stream:
  $f\ \ n =$
  $n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1,$

Patterns and Copatterns

$f\ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1,$

$$\begin{array}{lcl} f & : & \mathbb{N} \to \mathrm{Stream} \\ f & = & ? \end{array}$$

Patterns and Copatterns

$f\ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1,$

$$\begin{aligned} f &: \mathbb{N} \to \mathrm{Stream} \\ f &= \ ? \end{aligned}$$

Pattern match on $f : \mathbb{N} \to \mathrm{Stream}$:

$$\begin{aligned} f &: \mathbb{N} \to \mathrm{Stream} \\ f\ n &= \ ? \end{aligned}$$

Patterns and Copatterns

$f\ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1,$

$$\begin{array}{lcl} f : \mathbb{N} \to \mathrm{Stream} \\ f\ n & = & ? \end{array}$$

Copattern matching on $f\ n : \mathrm{Stream}$:

$$\begin{array}{lcl} f : \mathbb{N} \to \mathrm{Stream} \\ \mathrm{head}\ (f\ n) & = & ? \\ \mathrm{tail}\ \ (f\ n) & = & ? \end{array}$$

Patterns and Copatterns

$f\ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1,$

$$\begin{array}{lcl} f : \mathbb{N} \to \mathrm{Stream} & & \\ \mathrm{head}\ (f\ n) & = & ? \\ \mathrm{tail}\ \ (f\ n) & = & ? \end{array}$$

Pattern matching on the first $n : \mathbb{N}$:

$$\begin{array}{lcl} f : \mathbb{N} \to \mathrm{Stream} & & \\ \mathrm{head}\ (f\ 0) & = & ? \\ \mathrm{head}\ (f\ (\mathrm{S}\ n)) & = & ? \\ \mathrm{tail}\ \ (f\ n) & = & ? \end{array}$$

Patterns and Copatterns

$f\ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1,$

$$
\begin{array}{lcl}
f : \mathbb{N} \to \mathrm{Stream} & & \\
\mathrm{head}\ (f\ 0) & = & ? \\
\mathrm{head}\ (f\ (\mathrm{S}\ n)) & = & ? \\
\mathrm{tail}\ \ (f\ n) & = & ?
\end{array}
$$

Pattern matching on second $n : \mathbb{N}$:

$$
\begin{array}{lcl}
f : \mathbb{N} \to \mathrm{Stream} & & \\
\mathrm{head}\ (f\ 0) & = & ? \\
\mathrm{head}\ (f\ (\mathrm{S}\ n)) & = & ? \\
\mathrm{tail}\ \ (f\ 0) & = & ? \\
\mathrm{tail}\ \ (f\ (\mathrm{S}\ n)) & = & ?
\end{array}
$$

Patterns and Copatterns

$f\ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1,$

$$
\begin{array}{lcl}
f : \mathbb{N} \to \mathrm{Stream} & & \\
\mathrm{head}\ (f\ 0) & = & ? \\
\mathrm{head}\ (f\ (\mathrm{S}\ n)) & = & ? \\
\mathrm{tail}\ \ (f\ 0) & = & ? \\
\mathrm{tail}\ \ (f\ (\mathrm{S}\ n)) & = & ?
\end{array}
$$

Copattern matching on $\mathrm{tail}\ (f\ 0) : \mathrm{Stream}$

$$
\begin{array}{lcl}
f : \mathbb{N} \to \mathrm{Stream} & & \\
\mathrm{head}\ (f\ 0) & = & ? \\
\mathrm{head}\ (f\ (\mathrm{S}\ n)) & = & ? \\
\mathrm{head}\ (\mathrm{tail}\ (f\ 0)) & = & ? \\
\mathrm{tail}\ (\mathrm{tail}\ (f\ 0)) & = & ? \\
\mathrm{tail}\ (f\ (\mathrm{S}\ n)) & = & ?
\end{array}
$$

Patterns and Copatterns

$f\ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1,$

$$
\begin{array}{l}
f : \mathbb{N} \to \mathrm{Stream} \\
\mathrm{head}\ (f\ 0\ ) = \ ? \\
\mathrm{head}\ (f\ (\mathrm{S}\ n)) = \ ? \\
\mathrm{head}\ (\mathrm{tail}\ (f\ 0\ )) = \ ? \\
\mathrm{tail}\ (\mathrm{tail}\ (f\ 0\ )) = \ ? \\
\mathrm{tail}\ (f\ (\mathrm{S}\ n\ )) = \ ?
\end{array}
$$

Copattern matching on $\mathrm{tail}\ (f\ (\mathrm{S}\ n)) : \mathrm{Stream}$:

$$
\begin{array}{l}
f : \mathbb{N} \to \mathrm{Stream} \\
\mathrm{head}\ (f\ 0\ ) = \ ? \\
\mathrm{head}\ (f\ (\mathrm{S}\ n)) = \ ? \\
\mathrm{head}\ (\mathrm{tail}\ (f\ 0\ )) = \ ? \\
\mathrm{tail}\ (\mathrm{tail}\ (f\ 0\ )) = \ ? \\
\mathrm{head}\ (\mathrm{tail}\ (f\ (\mathrm{S}\ n))) = \ ? \\
\mathrm{tail}\ (\mathrm{tail}\ (f\ (\mathrm{S}\ n))) = \ ?
\end{array}
$$

Patterns and Copatterns

We resolve the goals:

$$
\begin{array}{lllcl}
\multicolumn{5}{l}{f : \mathbb{N} \to \mathrm{Stream}} \\
\mathrm{head} & & (f\ 0) & = & 0 \\
\mathrm{head} & (\mathrm{tail} & (f\ 0)) & = & 0 \\
\mathrm{tail} & (\mathrm{tail} & (f\ 0)) & = & f\ N \\
\mathrm{head} & & (f\ (\mathrm{S}\ n)) & = & \mathrm{S}\ n \\
\mathrm{head} & (\mathrm{tail} & (f\ (\mathrm{S}\ n))) & = & \mathrm{S}\ n \\
\mathrm{tail} & (\mathrm{tail} & (f\ (\mathrm{S}\ n))) & = & f\ n
\end{array}
$$

Results of paper in POPL

- Development of a recursive simply typed calculus (no termination check).
- Allows to derive schemata for pattern/copattern matching.
- Proof that subject reduction holds.

$$t : A, \quad t \longrightarrow t' \text{ implies } t' : A$$

Codata Type

- Idea of Codata Types:

$$\begin{array}{lcl} \multicolumn{3}{l}{\text{codata Stream : Setwhere}} \\ \quad\text{cons} & : & \mathbb{N} \to \text{Stream} \to \text{Stream} \end{array}$$

- Theoretical problem:
  Underlying assumption is

$$\forall s : \text{Stream}.\exists n, s'.s = \text{cons } n\ s'$$

  which results in undecidable equality.
- Results in Coq in a long known problem of subject reduction.
- In Agda severe restriction of elimination for coalgebras, which makes proving formulae involving coalgebras very difficult.

Problem of Subject reduction:

$$\begin{array}{l}
\text{data } _{==}_ \ \{A : \text{Set}\}\ (a : A) : A \to \text{Set where} \\
\quad \text{refl} : a == a \\
\\
\text{codata Stream} : \text{Set where} \\
\quad \text{cons} : \mathbb{N} \to \text{Stream} \to \text{Stream} \\
\\
\mathsf{zeros} : \text{Stream} \\
\mathsf{zeros} = \text{cons } 0\ \mathsf{zeros} \\
\\
\mathsf{force} : \text{Stream} \to \text{Stream} \\
\mathsf{force}\ s = \text{case } s \text{ of } (\text{cons } x\ y) \to \text{cons } x\ y \\
\\
\mathsf{lem1} : (s : \text{Stream}) \to s == \mathsf{force}(s)) \\
\mathsf{lem1}\ s = \text{case } s \text{ of } (\text{cons } x\ y) \to \text{refl} \\
\\
\mathsf{lem2} : \mathsf{zeros} == \text{cons } 0\ \mathsf{zeros} \\
\mathsf{lem2} = \mathsf{lem1}\ \mathsf{zeros} \\
\mathsf{lem2} \longrightarrow \text{refl} \ \ \mathsf{but}\ \neg(\text{refl} : \mathsf{zeros} == \text{cons } 0\ \mathsf{zeros})
\end{array}$$

Multiple Constructors in Algebras and Coalgebras

- Having more than one constructor in algebras correspond to disjoint union:

$$\begin{array}{l} \mathrm{data}\ \mathbb{N} : \mathrm{Set}\ \ \mathrm{where} \\ \quad \begin{array}{lcl} 0 & : & \mathbb{N} \\ \mathrm{S} & : & \mathbb{N} \to \mathbb{N} \end{array} \end{array}$$

  corresponds to

$$\begin{array}{l} \mathrm{data}\ \mathbb{N} : \mathrm{Set}\ \ \mathrm{where} \\ \quad \begin{array}{lcl} \mathrm{intro} & : & (1 + \mathbb{N}) \to \mathbb{N} \end{array} \end{array}$$

Multiple Constructors in Algebras and Coalgebras

- Dual of disjoint union is products, and therefore multiple destructors correspond to product:

$$\begin{array}{l} \mathrm{coalg}\ \mathrm{Stream} : \mathrm{Set}\ \mathrm{where} \\ \quad \begin{array}{lcl} \mathrm{head} & : & \mathrm{Stream} \to \mathbb{N} \\ \mathrm{tail} & : & \mathrm{Stream} \to \mathrm{Stream} \end{array} \end{array}$$

corresponds to

$$\begin{array}{l} \mathrm{coalg}\ \mathrm{Stream} : \mathrm{Set}\ \mathrm{where} \\ \quad \begin{array}{lcl} \mathrm{case} & : & \mathrm{Stream} \to (\mathbb{N} \times \mathrm{Stream}) \end{array} \end{array}$$

Codata Types Correspond to Disjoint Union

- Consider

$$\begin{array}{l} \text{codata coList} : \text{Set where} \\ \quad \begin{array}{lcl} \text{nil} & : & \text{coList} \\ \text{cons} & : & \mathbb{N} \to \text{coList} \to \text{coList} \end{array} \end{array}$$

- Cannot be simulated by a coalgebra with several destructors.

Simulating Codata Types by Simultaneous Algebras/Coalgebras

- Represent Codata as follows

mutual

$$
\begin{array}{l}
\quad \mathrm{coalg}\ \mathrm{coList} : \mathrm{Set}\ \mathrm{where} \\
\quad\quad \mathrm{unfold} : \mathrm{coList} \to \mathrm{coListShape} \\
\\
\quad \mathrm{data}\ \mathrm{coListShape} : \mathrm{Set}\ \mathrm{where} \\
\quad\quad \begin{array}{lcl}
\mathrm{nil} & : & \mathrm{coListShape} \\
\mathrm{cons} & : & \mathbb{N} \to \mathrm{coList} \to \mathrm{coListShape}
\end{array}
\end{array}
$$

Definition of Append

$$
\begin{array}{l}
\text{append} : \text{coList} \to \text{coList} \to \text{coList} \\
\text{append}\ l\ l' = ?
\end{array}
$$

Definition of Append

$$\begin{array}{l} \text{append} : \text{coList} \to \text{coList} \to \text{coList} \\ \text{append}\ l\ l' = ? \end{array}$$

We copattern match on $\text{append}\ l\ l' : \text{coList}$:

$$\begin{array}{l} \text{append} : \text{coList} \to \text{coList} \to \text{coList} \\ \text{unfold}\ (\text{append}\ l\ l') = ? \end{array}$$

Definition of Append

$$\begin{array}{l} \text{append} : \text{coList} \to \text{coList} \to \text{coList} \\ \text{unfold}\ (\text{append}\ l\ l') = ? \end{array}$$

We cannot pattern match on l.
But we can do so on $(\text{unfold}\ l)$:

$$\begin{array}{l} \text{append} : \text{coList} \to \text{coList} \to \text{coList} \\ \text{unfold}\ (\text{append}\ l\ l') = \\ \quad \text{case}\ (\text{unfold}\ l)\ \text{of} \\ \qquad \begin{array}{lcl} \text{nil} & \to & ? \\ (\text{cons}\ n\ l) & \to & ? \end{array} \end{array}$$

Definition of Append

$$
\begin{array}{l}
\mathrm{append} : \mathrm{coList} \to \mathrm{coList} \to \mathrm{coList} \\
\mathrm{unfold}\ (\mathrm{append}\ l\ l') = \\
\quad \mathrm{case}\ (\mathrm{unfold}\ l)\ \mathrm{of} \\
\qquad \begin{array}{lcl}
\mathrm{nil} & \to & ? \\
(\mathrm{cons}\ n\ l) & \to & ?
\end{array}
\end{array}
$$

We resolve the goals:

$$
\begin{array}{l}
\mathrm{append} : \mathrm{coList} \to \mathrm{coList} \to \mathrm{coList} \\
\mathrm{unfold}\ (\mathrm{append}\ l\ l') = \\
\quad \mathrm{case}\ (\mathrm{unfold}\ l)\ \mathrm{of} \\
\qquad \begin{array}{lcl}
\mathrm{nil} & \to & \mathrm{unfold}\ l' \\
(\mathrm{cons}\ n\ l) & \to & \mathrm{cons}\ n\ (\mathrm{append}\ l\ l')
\end{array}
\end{array}
$$

Fibonacci Numbers

Efficient Haskell version adapted to our codata notation:

$$
\begin{array}{l}
\mathrm{codata\ Stream} : \mathrm{Set\ where} \\
\quad \mathrm{cons} : \mathbb{N} \to \mathrm{Stream} \to \mathrm{Stream}
\end{array}
$$

$$
\begin{array}{l}
\mathrm{tail} : \mathrm{Stream} \to \mathrm{Stream} \\
\mathrm{tail}\ (\mathrm{cons}\ n\ l) = l
\end{array}
$$

$$
\begin{array}{l}
\mathrm{addStream} : \mathrm{Stream} \to \mathrm{Stream} \to \mathrm{Stream} \\
\mathrm{addStream}\ (\mathrm{cons}\ n\ l)\ (\mathrm{cons}\ n'\ l') = \mathrm{cons}\ (n + n')\ (\mathrm{addStream}\ l\ l')
\end{array}
$$

$$
\begin{array}{l}
\mathrm{fib} : \mathrm{Stream} \\
\mathrm{fib} = \mathrm{cons}\ 1\ (\mathrm{cons}\ 1\ (\mathrm{addStream}\ \mathrm{fib}\ (\mathrm{tail}\ \mathrm{fib})))
\end{array}
$$

Requires lazy evaluation.

Fibonacci Numbers using Coalgebras

$$
\begin{array}{l}
\text{coalg Stream} : \text{Set where} \\
\quad \begin{array}{lcl} \text{head} & : & \text{Stream} \to \mathbb{N} \\ \text{tail} & : & \text{Stream} \to \text{Stream} \end{array}
\end{array}
$$

$$
\begin{array}{l}
\text{addStream} : \text{Stream} \to \text{Stream} \to \text{Stream} \\
\begin{array}{lcl}
\text{head } (\text{addStream } l\ l') & = & \text{head } l + \text{head } l' \\
\text{tail } \ (\text{addStream } l\ l') & = & \text{addStream } (\text{tail } l)\ (\text{tail } l')
\end{array}
\end{array}
$$

$$
\begin{array}{l}
\text{fib} : \text{Stream} \\
\begin{array}{lcl}
\text{head fib} & = & 1 \\
\text{head } (\text{tail fib}) & = & 1 \\
\text{tail } \ (\text{tail fib}) & = & \text{addStream fib } (\text{tail fib})
\end{array}
\end{array}
$$

No laziness required. Requires full corecursion (but terminates).

Conclusion

- Symmetry between
  - algebras and coalgebras,
  - iteration and coiteration,
  - recursion and corecursion,
  - patterns and copatterns.
- Unknown: dual of induction (requires codependent types?)
- Codata construct assumes every element is introduced by a constructor, which results in
  - either undecidable equality
  - or requires sophisticated restrictions on reduction rule which are difficult to get right.
    - Problem of subreduction in Coq.
    - Overly restriction on elimination in Agda.
- Weakly final coalgebras solve this problem, but add small overhead when programming.