Coinduction, Corecursion, Copatterns

Anton Setzer

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

Lisbon, 31 January 2013

Algebraic Data Types

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

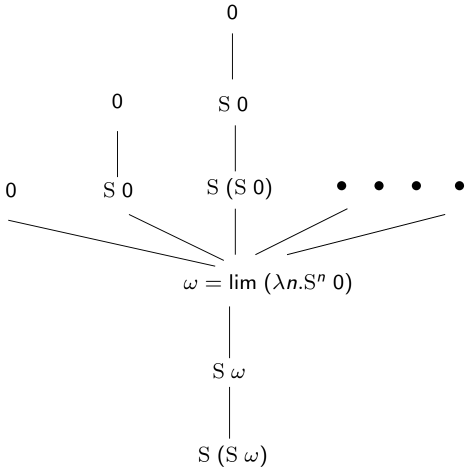
$$
\begin{array}{l}
\mathrm{data}\ \mathbb{N} : \mathrm{Set}\ \ \mathrm{where} \\
\quad \begin{array}{lcl}
\mathsf{0} & : & \mathbb{N} \\
\mathrm{S} & : & \mathbb{N} \to \mathbb{N}
\end{array}
\end{array}
$$

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

Algebraic Data Types

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

Kleene's O



Algebraic Data Types as F-Algebras

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

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

$$F(X) := 1 + X$$

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

Algebraic Data Types as F-Algebras

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

can be with

$$F(X) := 1 + (\mathbb{N} \times X)$$

rewritten as

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

Algebraic Data Types as F-Algebras

Finally

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

can be with

$$F(X) := 1 + X + (\mathbb{N} \to X)$$

rewritten as

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

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 $\mathbb{N}$):

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

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

So with $a_0 := f(\mathrm{inl})$ and $f_0(x) := f(\mathrm{inr}(x))$

$$g(n) = f_0^n(a_0)$$

Recursion

The principle of recursion can be derived using uniqueness:
Assume

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

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 e.g.

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

Recursion

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

Define $f : (1 + (\mathbb{N} \times A)) \to (\mathbb{N} \times A)$

$$
\begin{array}{lcl}
f(\mathrm{inl}) & = & (0, a_0) \\
f(\mathrm{inr}(n, a)) & = & (\mathrm{S}(n), f_0(n, a))
\end{array}
$$

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

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

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

Induction

Induction can be regarded as dependent elimination:
Assume

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

We derive $g : (n : \mathbb{N}) \to A(n)$ 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.

Coalgebras

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

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

Coalgebras

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

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

Let

$$\mathrm{case}(s) \quad = \quad (\mathrm{head}(s), \mathrm{tail}(s))$$

and

$$f(a) = (f_0(a), f_1(a))$$

Guarded Recursion

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

Resulting equations:

$$\begin{array}{lcl} \mathrm{head}(g(a)) & = & \pi_0(\mathrm{case}(g(a))) = \pi_0(f_0(a), g(f_1(a))) = f_0(a) \\ \mathrm{tail}(g(a)) & = & \pi_1(\mathrm{case}(g(a))) = \pi_1(f_0(a), g(f_1(a))) = g(f_1(a)) \end{array}$$

Example of Guarded Recursion

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

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:

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

Corecursion

Corecursion allows for tail to escape into a stream.
Example:

$$\begin{array}{rcl} \multicolumn{3}{l}{\mathrm{cons} : (\mathbb{N} \times \mathrm{Stream}) \to \mathrm{Stream}} \\ \mathrm{head}(\mathrm{cons}(n, s)) & = & n \\ \mathrm{tail}(\ \mathrm{cons}(n, s)) & = & s \end{array}$$

Corecursion

More generally, if

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

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

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

Corecursion vs Recursion

Compare with recursion which allowed for

$$
\begin{array}{lcl}
A & : & \mathrm{Set} \\
a_0 & : & A \\
f_0 & : & (\mathbb{N} \times A) \to A
\end{array}
$$

To define

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

Nested Corecursion

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

Even more general schemas can be defined.

Weakly Final Coalgebra

- Equality for final coalgebras is undecidable:
  Two streams

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

  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:

$$f(n) = n, n, n{-}1, n{-}1, \ldots 0, 0, N, N, N{-}1, N{-}1, \ldots 0, 0, N, N, N{-}1, N{-}$$

- Step 1:

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

- Step 2: Apply f to variable:

$$f(n) \quad = \quad ?$$

- Step 3: Make **copattern matching** on $f(n) : \mathrm{Stream}$:

$$\begin{array}{lcl} \mathrm{head}(f(n)) & = & ? \\ \mathrm{tail}(\ \ f(n)) & = & ? \end{array}$$

Patterns and Copatterns

- We make **pattern matching** on $n = 0$, $n = \mathrm{S}(m)$:

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

Patterns and Copatterns

$$f(n) = n, n, n{-}1, n{-}1, \ldots 0, 0, N, N, N{-}1, N{-}1, \ldots 0, 0, N, N, N{-}1, N{-}1, \ldots$$

- We make **copattern matching** on Stream:

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

Results of paper in POPL

- Development of a recursive simply typed calculus (no termination check).
- Allows to derive schemas 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:

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

- Theoretical problem:
  Underlying assumption is

  $$\forall s : \mathrm{Stream}.\exists n, s'.s = \mathrm{cons}(n, s')$$

  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 formulas involving coalgebras very difficult.

Problem of Subject reduction

$\text{data } \_{==}\_ \ \{A : \text{Set}\}\ (a : A) : A \to \text{Set where}$
$\quad \text{refl} : a == a$

$\text{codata Stream} : \text{Set where}$
$\quad \text{cons} : (\mathbb{N} \times \text{Stream}) \to \text{Stream}$

$\mathsf{zeros} : \text{Stream}$
$\mathsf{zeros} = \text{cons}(0, \mathsf{zeros})$

$\mathsf{force} : \text{Stream} \to \text{Stream}$
$\mathsf{force}(s) = \text{case } s \text{ of cons}(x, y) \to \text{cons}(x, y)$

$\mathsf{lem1} : (s : \text{Stream}) \to s == \mathsf{force}(s))$
$\mathsf{lem1}(s) = \text{case } s \text{ of cons}(x, y) \to \text{refl}$

$\mathsf{lem2} : \mathsf{zeros} == \text{cons}(0, \mathsf{zeros})$
$\mathsf{lem2} = \mathsf{lem1}(\mathsf{zeros}) \longrightarrow \text{refl} \quad \neg(\text{refl} : \mathsf{zeros} == \text{cons}(0, \mathsf{zeros}))$

Multiple Constructors in Algebras and Coalgebras

- Several constructors in algebras correspond to disjoint union:

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

  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

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

- Cannot be simulated by a coalgebra with several destructors.

Simulating Codata Types by Simultaneous Algebras/Coalgebras

- Represent Codata as follows

mutual

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

Example Append

$$\begin{array}{l}\mathrm{append} : (\mathrm{coList} \times \mathrm{coList}) \to \mathrm{coList} \\ \mathrm{append}(l, l') = \mathrm{appendaux}(\mathrm{unfold}(l), l')\end{array}$$

$$\begin{array}{lcl}\multicolumn{3}{l}{\mathrm{appendaux} : (\mathrm{coListShape} \times \mathrm{coList}) \to \mathrm{coList}} \\ \mathrm{appendaux}(\mathrm{nil}, l') & = & l' \\ \mathrm{unfold}(\mathrm{appendaux}(\mathrm{cons}(n, l), l')) & = & \mathrm{cons}(n, \mathrm{append}(l, l'))\end{array}$$

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 undecidable equality.
- Weakly final coalgebras solves this problem, but adds small overhead when programming.