Copatterns: Programming Infinite Structures by Observations

Andreas Abel (LMU Munich)
Talk given and Slides prepared by Anton Setzer (Swansea, UK)

(Joint work of Andreas Abel, Brigitte Pientka, Anton Setzer, David Thibodeau)

Types 2013 Toulouse, Wed, 24 April 2013

Coalgebras in Functional Programming

- Originally functional programming based on
  - function types,
  - inductive data types.
- In computer science, many computations are interactive.
- Since interactions might go on forever (if not terminated by the user), they correspond to non-wellfounded data types
  - Streams, which are infinite lists,
  - non-wellfounded trees (IO-trees).

Codata Type

- Idea of Codata Types:

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

- Same definition as inductive data type but we are allowed to have infinite chains of constructors

$$\text{cons}\ n_0\ (\text{cons}\ n_1\ (\text{cons}\ n_2\ \cdots))$$

- **Problem 1:** Non-normalisation.
- **Problem 2:** Equality between streams is equality between all elements, and therefore undecidable.
- **Problem 3:** Underlying assumption is

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

which results in undecidable equality.

Subject Reduction Problem

- In order to repair problem of normalisation restrictions on reductions were introduced.
- Resulted in Coq in a long known problem of **subject reduction**.
- In order to avoid this, in Agda dependent elimination for coalgebras disallowed.
  - Makes it difficult to use.

Problem of Subject reduction:

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

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

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

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

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

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

Coalgebraic Formulation of Coalgebras

- Solution is to follow the long established categorical formulation of coalgebras.

Initial F-Algebras

- Inductive data types correspond to initial F-Algebras.
- E.g. the natural numbers can be formulated as

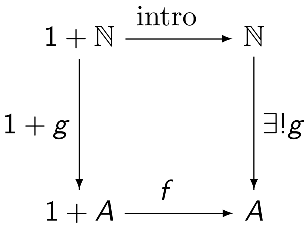
$$
\begin{array}{l}
F(X) = 1 + X \\
\mathrm{intro} : F(\mathbb{N}) \to \mathbb{N} \\
\mathrm{intro}\ (\mathrm{inl}\ *) \quad = \quad 0 \\
\mathrm{intro}\ (\mathrm{inl}\ n) \quad = \quad \mathrm{S}\ n
\end{array}
$$

and we get the diagram

$$
\begin{array}{ccccc}
1 + \mathbb{N} & = & F(\mathbb{N}) & \xrightarrow{\mathrm{intro}} & \mathbb{N} \\
 & 1 + g = F(g) & \downarrow & & \downarrow\ \exists! g \\
1 + A & = & F(A) & \xrightarrow{f} & A
\end{array}
$$

Iteration

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



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

By choosing arbitrary f we can define g by pattern matching on its argument n:

$$
\begin{array}{lll}
g\ 0 & = & a_0 \\
g\ (\mathrm{S}\ n) & = & f\ (g\ n) \text{ for some } f : \mathbb{N} \to \mathbb{N}
\end{array}
$$

Recursion and Induction

- From the principle of unique iteration one can derive the principle of recursion:
  Assume

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

  We can then define $g : \mathbb{N} \to A$ s.t.

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

- Induction is as recursion but now

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

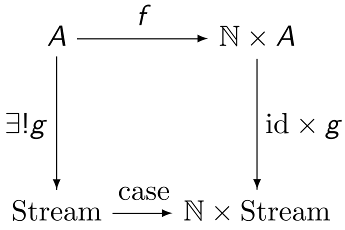
Coalgebras

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

$$\begin{array}{ccc} A & \xrightarrow{\;f\;} & \mathrm{F}(A) \\ {\scriptstyle \exists! g}\downarrow & & \downarrow{\scriptstyle \mathrm{F}(g)} \\ \mathrm{F}^\infty & \xrightarrow{\;\mathrm{case}\;} & \mathrm{F}(\mathrm{F}^\infty) \end{array}$$

Coalgebras

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



Let

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

and

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

Guarded Recursion

$$
\begin{array}{ccc}
A & \xrightarrow{\langle f_0, f_1 \rangle} & \mathbb{N} \times A \\
\exists! g \downarrow & & \downarrow \mathrm{id} \times g \\
\mathrm{Stream} & \xrightarrow{\langle \mathrm{head}, \mathrm{tail} \rangle} & \mathbb{N} \times \mathrm{Stream}
\end{array}
$$

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

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

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:

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

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

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

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

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

Definition of cons by Corecursion

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

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

Nested Corecursion

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

Even more general schemata can be defined.

Definition of Coalgebras by Observations

- We see now that elements of coalgebras are defined by their observations:
  An element s of Stream is given by defining

$$\begin{array}{lll} \mathrm{head} & s & : & \mathbb{N} \\ \mathrm{tail} & s & : & \mathrm{Stream} \end{array}$$

- This generalises the function type.
  Functions $f : A \to B$ are as well determined by observations, namely by defining

$$f\ a : B$$

  - An $f : A \to B$ is any program which applied to $a : A$ returns some $b : B$.
- **Inductive data types** are defined by **construction** **coalgebraic data types** and **functions** by **observations**.

Relationship to Objects in Object-Oriented Programming

- Objects in Object-Oriented Programming are types which are defined by their observations.
- Therefore objects are coalgebraic types by nature.

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

Patterns and Copatterns

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

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

Patterns and Copatterns

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

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

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

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

Patterns and Copatterns

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

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

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

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

Patterns and Copatterns

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

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

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

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

Patterns and Copatterns

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

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

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

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

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

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

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

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

Patterns and Copatterns

We resolve the goals:

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

Results of paper in POPL (2013)

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

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

Fibonacci Numbers

Efficient Haskell version adapted to our codata notation:

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

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

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

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

Requires lazy evaluation.

Fibonacci Numbers using Coalgebras

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

$$\begin{array}{lll}
\multicolumn{3}{l}{\text{addStream} : \text{Stream} \to \text{Stream} \to \text{Stream}} \\
\text{head (addStream } l\ l') & = & \text{head } l + \text{head } l' \\
\text{tail } \ \text{(addStream } l\ l') & = & \text{addStream (tail } l\text{) (tail } l'\text{)}
\end{array}$$

$$\begin{array}{lll}
\multicolumn{3}{l}{\text{fib} : \text{Stream}} \\
\text{head fib} & = & 1 \\
\text{head (tail fib)} & = & 1 \\
\text{tail } \ \text{(tail fib)} & = & \text{addStream fib (tail fib)}
\end{array}$$

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

Multiple Constructors in Algebras and Coalgebras

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

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

  corresponds to

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

Multiple Constructors in Algebras and Coalgebras

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

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

corresponds to

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

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

$$
\begin{array}{l}
\textsf{mutual} \\
\quad \text{coalg coList} : \text{Set where} \\
\quad\quad \text{unfold} : \text{coList} \to \text{coListShape} \\
\\
\quad \text{data coListShape} : \text{Set where} \\
\quad\quad \text{nil} \quad : \quad \text{coListShape} \\
\quad\quad \text{cons} \quad : \quad \mathbb{N} \to \text{coList} \to \text{coListShape}
\end{array}
$$

Definition of Append

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

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

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

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

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

We resolve the goals:

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

Conclusion

- Symmetry between
  - algebras and coalgebras,
  - iteration and coiteration,
  - recursion and corecursion,
  - patterns and copatterns.
- Final algebras are defined by constuction,
  coalegbras and function types by observation.
- 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.
    - Too restrictive elimination principle in Agda.
- Weakly final coalgebras solve this problem, by having reduction rules which can always be applied independent of context.

More Details

- More details can be found in my proper talk tomorrow.
  - Assumption $\forall s : \mathrm{Stream}.\exists n, s'.s = \mathrm{cons}\ n\ s'$ results in undecidable equality.
  - How to replace copattern matching by combinators.