The Dual of Pattern Matching - Copattern Matching

Anton Setzer
Swansea, UK

(Joint work with Andreas Abel, Brigitte Pientka, and David Thibodeau)

Birmingham Seminar, 22 November 2013

From Codata to Coalgebras

Algebras and Coalgebras

Patterns and Copatterns

Codata types and Decidable Equality

Reduction of Mixed Pattern/Copattern Matching to Operators

Conclusion

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:

$$
\begin{array}{l}
\text{data } \_{==}\_ \ \{A : \mathrm{Set}\}\ (a : A) : A \to \mathrm{Set} \text{ where} \\
\quad \mathrm{refl} : a == a
\end{array}
$$

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

$$
\begin{array}{l}
\mathsf{zeros} : \mathrm{Stream} \\
\mathsf{zeros} = \mathrm{cons}\ 0\ \mathsf{zeros}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{force} : \mathrm{Stream} \to \mathrm{Stream} \\
\mathsf{force}\ s = \mathrm{case}\ s\ \mathrm{of}\ (\mathrm{cons}\ x\ y) \to \mathrm{cons}\ x\ y
\end{array}
$$

$$
\begin{array}{l}
\mathsf{lem1} : (s : \mathrm{Stream}) \to s == \mathsf{force}(s)) \\
\mathsf{lem1}\ s = \mathrm{case}\ s\ \mathrm{of}\ (\mathrm{cons}\ x\ y) \to \mathrm{refl}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{lem2} : \mathsf{zeros} == \mathrm{cons}\ 0\ \mathsf{zeros} \\
\mathsf{lem2} = \mathsf{lem1}\ \mathsf{zeros} \\
\mathsf{lem2} \longrightarrow \mathrm{refl} \quad \text{but } \neg(\mathrm{refl} : \mathsf{zeros} == \mathrm{cons}\ 0\ \mathsf{zeros})
\end{array}
$$

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

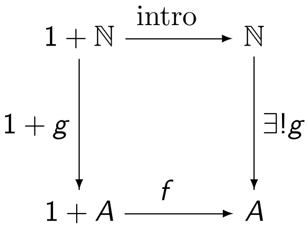
$$
\begin{array}{l}
F(X) = 1 + X \\
\mathrm{intro} : F(\mathbb{N}) \rightarrow \mathbb{N} \\
\mathrm{intro}\ (\mathrm{inl}\ *) \quad = \quad 0 \\
\mathrm{intro}\ (\mathrm{inl}\ n) \quad = \quad \mathrm{S}\ n
\end{array}
$$

and we get the diagram

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

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:

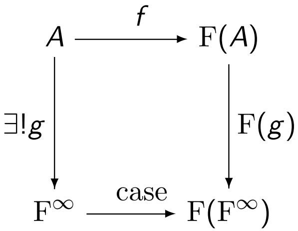
$$
\begin{array}{lll}
g\ 0 & = & a_0 \\
g\ (\mathrm{S}\ n) & = & f\ (g\ n) \text{ for some } f : \mathbb{N} \to \mathbb{N}
\end{array}
$$

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:



Coalgebras

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

$$\begin{array}{ccc} A & \xrightarrow{\quad f \quad} & \mathbb{N} \times A \\ {\scriptstyle \exists ! g}\downarrow & & \downarrow{\scriptstyle \mathrm{id} \times g} \\ \mathrm{Stream} & \xrightarrow{\mathrm{case}} & \mathbb{N} \times \mathrm{Stream} \end{array}$$

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

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

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

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

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

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.

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

Definition of cons by Corecursion

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

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

Nested Corecursion

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

Even more general schemata can be defined.

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

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

  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

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

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

Copattern matching on $f : \mathbb{N} \to \mathrm{Stream}$:

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

Patterns and Copatterns

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

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

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

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

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

Solve first case, copattern match on second case:

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

Patterns and Copatterns

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

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

Solve second line, pattern match on n

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

Patterns and Copatterns

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

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

Solve remaining cases

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

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.

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

Theorem Regarding Undecidabilty of Equality

Theorem

Assume the following:

- *There exists a subset* $\mathrm{Stream} \subseteq \mathbb{N}$,
- *computable functions*
  $\mathrm{head} : \mathrm{Stream} \to \mathbb{N}$, $\mathrm{tail} : \mathrm{Stream} \to \mathrm{Stream}$,
- *a decidable equality* $\_ == \_$ *on* Stream *which is congruence,*
- *the possibilty to define elements of* Stream *by guarded recursion based on primitive recursive functions* $f, g : \mathbb{N} \to \mathbb{N}$, *such that the standard equalities related to guarded recursion hold.*

Then it is not possible to fulfil the following condition:

$$\forall s, s' : \mathrm{Stream}.\mathrm{head}\ s = \mathrm{head}\ s' \wedge \mathrm{tail}\ s == \mathrm{tail}\ s' \to s == s' \qquad (*)$$

Consequences for Codata Approach

Remark

Condition $(*)$ *is fulfilled if we have an operation*
$\mathrm{cons} : \mathbb{N} \to \mathrm{Stream} \to \mathrm{Stream}$ *preserving equalities s.t.*

$$\forall s : \mathrm{Stream}. s = \mathrm{cons}\ (\mathrm{head}\ s)\ (\mathrm{tail}\ s)$$

So we cannot have a type theory with streams, decidable type checking and decidable equality on streams such that

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

as assumed by the codata approach.

Proof of Theorem

- Assume we had the above.
- By

$$s \approx n_0 :: n_1 :: n_2 :: \cdots n_k :: s'$$

  we mean the equations using head, tail expressing that s behaves as the stream indicated on the right hand side.
- Define by guarded recursion $l : \mathrm{Stream}$

$$l \approx 1 :: 1 :: 1 :: \cdots$$

Proof of Theorem

- For e code for a Turing machine define by guarded recursion based on primitive recursion functions f, g s.t. if e terminates after n steps and returns result k then

$$f\ e \approx \underbrace{0 :: 0 :: 0 :: \cdots :: 0}_{n \text{ times}} :: l$$

$$g\ e \approx \begin{cases} \underbrace{0 :: 0 :: 0 :: \cdots :: 0}_{n \text{ times}} :: l & \text{if } k = 0 \\ \underbrace{0 :: 0 :: 0 :: \cdots :: 0}_{n+1 \text{ times}} :: l & \text{if } k > 0 \end{cases}$$

Proof of Theorem

$$f\ e \quad \approx \quad \underbrace{0 :: 0 :: 0 :: \cdots :: 0}_{n \text{ times}} :: l$$

$$g\ e \quad \approx \quad \begin{cases} \underbrace{0 :: 0 :: 0 :: \cdots :: 0}_{n \text{ times}} :: l & \text{if } k = 0 \\ \underbrace{0 :: 0 :: 0 :: \cdots :: 0}_{n+1 \text{ times}} :: l & \text{if } k > 0 \end{cases}$$

- If e terminates after n steps with result 0 then

$$f\ e == g\ e$$

- If e terminates after n steps with result > 0 then

$$\neg(f\ e == g\ e)$$

Proof of Theorem

- So

$$\lambda e.(f\ e == g\ e)$$

  separates the TM with result 0 from those with result > 0.
- But these two sets are inseparable.

Operators for Primitive (Co)Recursion

$$
\begin{array}{lcl}
\mathrm{P}_{\mathbb{N},A} : A \to (\mathbb{N} \to A \to A) \to \mathbb{N} \to A \\
\mathrm{P}_{\mathbb{N},A}\ \mathrm{step}_0\ \mathrm{step}_\mathrm{S}\ 0 & = & \mathrm{step}_0 \\
\mathrm{P}_{\mathbb{N},A}\ \mathrm{step}_0\ \mathrm{step}_\mathrm{S}\ (\mathrm{S}\ n) & = & \mathrm{step}_\mathrm{S}\ n\ (\mathrm{P}_{\mathbb{N},A}\ \mathrm{step}_0\ \mathrm{step}_\mathrm{S}\ n)
\end{array}
$$

$$
\begin{array}{lcl}
\mathrm{coP}_{\mathrm{Stream},A} : (A \to \mathbb{N}) \to (A \to (\mathrm{Stream} + A)) \to A \to \mathrm{Stream} \\
\mathrm{head}\ (\mathrm{coP}_{\mathrm{Stream},A}\ \mathrm{step}_\mathrm{head}\ \mathrm{step}_\mathrm{tail}\ a) & = & \mathrm{step}_\mathrm{head}\ a \\
\mathrm{tail}\ \ \ (\mathrm{coP}_{\mathrm{Stream},A}\ \mathrm{step}_\mathrm{head}\ \mathrm{step}_\mathrm{tail}\ a) & = & \\
\quad \mathrm{case}_{\mathrm{Stream},A,\mathrm{Stream}}\ \mathrm{id}\ (\mathrm{coP}_{\mathrm{Stream},A}\ \mathrm{step}_\mathrm{head}\ \mathrm{step}_\mathrm{tail})\ (\mathrm{step}_\mathrm{tail}\ a)
\end{array}
$$

Operators for full/primitive (co)recursion

$$\mathrm{R}_{\mathbb{N},A} : ((\mathbb{N} \to A) \to A) \to ((\mathbb{N} \to A) \to \mathbb{N} \to A) \to \mathbb{N} \to A$$

$$\begin{array}{lcl} \mathrm{R}_{\mathbb{N},A}\ \mathrm{step}_0\ \mathrm{step}_\mathrm{S}\ 0 & = & \mathrm{step}_0\ (\mathrm{R}_{\mathbb{N},A}\ \mathrm{step}_0\ \mathrm{step}_\mathrm{S}) \\ \mathrm{R}_{\mathbb{N},A}\ \mathrm{step}_0\ \mathrm{step}_\mathrm{S}\ (\mathrm{S}\ n) & = & \mathrm{step}_\mathrm{S}\ (\mathrm{R}_{\mathbb{N},A}\ \mathrm{step}_0\ \mathrm{step}_\mathrm{S})\ n \end{array}$$

$$\begin{array}{ll} \mathrm{coR}_{\mathrm{Stream},A} : & ((A \to \mathrm{Stream}) \to A \to \mathbb{N}) \\ & \to ((A \to \mathrm{Stream}) \to A \to \mathrm{Stream}) \\ & \to \mathrm{Stream} \end{array}$$

$$\begin{array}{lcl} \mathrm{head}\ (\mathrm{coR}_{\mathrm{Stream},A}\ \mathrm{step}_\mathrm{head}\ \mathrm{step}_\mathrm{tail}\ a) & = & \mathrm{step}_\mathrm{head} \\ & & (\mathrm{coR}_{\mathrm{Stream},A}\ \mathrm{step}_\mathrm{head}\ \mathrm{step}_\mathrm{tail})\ a \\ \mathrm{tail}\ (\mathrm{coR}_{\mathrm{Stream},A}\ \mathrm{step}_\mathrm{head}\ \mathrm{step}_\mathrm{tail}\ a) & = & \mathrm{step}_\mathrm{tail} \\ & & (\mathrm{coR}_{\mathrm{Stream},A}\ \mathrm{step}_\mathrm{head}\ \mathrm{step}_\mathrm{tail})\ a \end{array}$$

Consider Example from above

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

This example can be reduced to primitive (co)recursion.
Step 1: Following the development of the (co)pattern matching definition, unfold it into simulteneous non-nested (co)pattern matching definitions.

Step 1: Unnesting of Nested (Co)Pattern Matching

We follow the steps in the pattern matching:
We start with

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

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

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

corresponds to

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

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

Pattern matching on tail (tail (f n)):

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

corresponds to

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

Step 2: Reduction to Primitive (Co)recursion

- This can now easily be reduced to full (co)recursion.
- In this example we can reduce it to primitive (co)recursion.
- First combine f, g into one function $f + g$.

$$
\begin{aligned}
&f : \mathbb{N} \to \mathrm{Stream} \\
&f\ n && = && (f+g)\ (\underline{\mathrm{f}}\ n) \\
\\
&(f+g) : (\underline{\mathrm{f}}(\mathbb{N}) + \underline{\mathrm{g}}(\mathbb{N})) \to \mathrm{Stream} \\
&\mathrm{head} \quad ((f+g)\ (\underline{\mathrm{f}}\ n)) && = && n \\
&\mathrm{head} \quad ((f+g)\ (\underline{\mathrm{g}}\ n)) && = && n \\
&\mathrm{tail} \quad ((f+g)\ (\underline{\mathrm{f}}\ n)) && = && (f+g)\ (\underline{\mathrm{g}}\ n) \\
&\mathrm{tail} \quad ((f+g)\ (\underline{\mathrm{f}}\ n)) && = && k\ n \\
\\
&k : \mathbb{N} \to \mathrm{Stream} \\
&k\ 0 && = && (f+g)\ (\underline{\mathrm{f}}\ N) \\
&k\ (\mathrm{S}\ n) && = && (f+g)\ (\underline{\mathrm{f}}\ n)
\end{aligned}
$$

Unfolding of the Pattern Matchings

- The call of k has result always of the form $(f + g)(\ fbf\ \ n))$.
  So we can replace the recursive call $k\ n$ by $(f + g)(\underline{\mathrm{f}}\ (k'\ n))$.

$$
\begin{aligned}
&f : \mathbb{N} \to \mathrm{Stream} \\
&f\ n && = && (f+g)\ (\underline{\mathrm{f}}\ n) \\
\\
&(f+g) : (\underline{\mathrm{f}}(\mathbb{N}) + \underline{\mathrm{g}}(\mathbb{N})) \to \mathrm{Stream} \\
&\mathrm{head}\quad ((f+g)\ (\underline{\mathrm{f}}\ n)) && = && n \\
&\mathrm{head}\quad ((f+g)\ (\underline{\mathrm{g}}\ n)) && = && n \\
&\mathrm{tail}\quad ((f+g)\ (\underline{\mathrm{f}}\ n)) && = && (f+g)\ (\underline{\mathrm{g}}\ n) \\
&\mathrm{tail}\quad ((f+g)\ (\underline{\mathrm{f}}\ n)) && = && (f+g)\ (\underline{\mathrm{f}}\ (k'\ n)) \\
\\
&k' : \mathbb{N} \to \mathbb{N} \\
&k\ 0 && = && N \\
&k\ (\mathrm{S}\ n) && = && n
\end{aligned}
$$

Unfolding of the Pattern Matchings

- $(f + g)$ can be defined by primitive corecursion.
- k' can be defined by primitive recursion.

$$
\begin{aligned}
&f : \mathbb{N} \to \mathrm{Stream}\\
&f\ n \quad = \quad (f+g)\ (\underline{\mathrm{f}}\ n)
\end{aligned}
$$

$$
\begin{aligned}
&(f+g) : (\underline{\mathrm{f}}(\mathbb{N}) + \underline{\mathrm{g}}(\mathbb{N})) \to \mathrm{Stream}\\
&(f+g) =\\
&\quad \mathrm{coP}_{\mathrm{Stream},(\underline{\mathrm{f}}(\mathbb{N})+\underline{\mathrm{g}}(\mathbb{N}))}\ (\lambda x.\mathrm{case}_r(x)\ \mathrm{of}\\
&\qquad\qquad (\underline{\mathrm{f}}\ n) \quad \longrightarrow \quad n\\
&\qquad\qquad (\underline{\mathrm{g}}\ n) \quad \longrightarrow \quad n)\\
&\quad\ (\lambda x.\mathrm{case}_r(x)\ \mathrm{of}\\
&\qquad\qquad (\underline{\mathrm{f}}\ n) \quad \longrightarrow \quad \underline{\mathrm{g}}\ n\\
&\qquad\qquad (\underline{\mathrm{g}}\ n) \quad \longrightarrow \quad \underline{\mathrm{f}}\ (k'\ n))
\end{aligned}
$$

$$
\begin{aligned}
&k' : \mathbb{N} \to \mathbb{N}\\
&k' = \mathrm{P}_{\mathbb{N},\mathbb{N}}\ N\ (\lambda n, ih.n)
\end{aligned}
$$

Reduction to Primitive (Co)Recursion

- The case distinction can be trivially replaced by the case distinction operator.

$$f : \mathbb{N} \to \mathrm{Stream}$$
$$f\ n \quad = \quad (f + g)\ (\underline{\mathrm{f}}\ n)$$

$$(f + g) : (\underline{\mathrm{f}}(\mathbb{N}) + \underline{\mathrm{g}}(\mathbb{N})) \to \mathrm{Stream}$$
$$\begin{aligned}(f + g) =&\\ &\mathrm{coP}_{\mathrm{Stream}, \underline{\mathrm{f}}(\mathbb{N}) + \underline{\mathrm{g}}(\mathbb{N})}\ (\mathrm{case}_{\underline{\mathrm{f}}(\mathbb{N}) + \underline{\mathrm{g}}(\mathbb{N})}\ \mathrm{id}\ \mathrm{id})\\ &\qquad (\mathrm{case}_{\underline{\mathrm{f}}(\mathbb{N}) + \underline{\mathrm{g}}(\mathbb{N})}\ \underline{\mathrm{g}}\ (\underline{\mathrm{f}} \circ k'))\end{aligned}$$

$$k' : \mathbb{N} \to \mathbb{N}$$
$$k' = \mathrm{P}_{\mathbb{N}, \mathbb{N}}\ N\ (\lambda n, ih.n)$$

Conclusion

- Codata types make the assumption

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

  which cannot be combined with a decidable equality.
- In general Codata types cause problems such as subject reduction.
- Solution:
  - Coalgebra are determined by their elimination rule.
  - Introduction rule corresponds to copattern matching.
- Solves problem of subject reduction.

Conclusion

- One can reduce certain cases of recursive nested (co)pattern matching to primitive (co)recursion.
  - Systematic treatment needs still to be done.
  - Cases which can be reduced should be those to be accepted by a termination checker.
  - If the reduction succeeds we get a normalising version (by Mendler and Geuvers).
  - Therefore a termination checked version of the calculus is normalising.