Coinduction, Corecursion, Copatterns

Anton Setzer

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

Swansea, 21 February 2013
Initial Algebras in Functional Programming

Coalgebras and Copatterns

Codata Types

Conclusion
Initial Algebras in Functional Programming

Coalgebras and Copatterns

Codata Types

Conclusion
Algebraic Data Types

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

```haskell
data N : Set where
  0 : N
  S : N → N

data NatList : Set where
  nil : NatList
  cons : N → NatList → NatList
```
data \mathbb{N} : \text{Set} \text{ where}
\begin{align*}
0 & : \mathbb{N} \\
S & : \mathbb{N} \rightarrow \mathbb{N}
\end{align*}
can be rewritten as

\begin{align*}
data \mathbb{N} : \text{Set} \text{ where} \\
\text{intro} & : (1 + \mathbb{N}) \rightarrow \mathbb{N}
\end{align*}

or with \( F(X) := 1 + X \)

\begin{align*}
data \mathbb{N} : \text{Set} \text{ where} \\
\text{intro} & : F(\mathbb{N}) \rightarrow \mathbb{N}
\end{align*}

So \( 0 = \text{intro \, inl} \) and \( S \, n = \text{intro \, (inr \, n)} \).
data NatList : Set where
  nil : NatList
  cons : ℕ → NatList → NatList

can we written as

data NatList : Set where
  nil : 1 → NatList
  cons : (ℕ × NatList) → NatList

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

data NatList : Set where
  intro : F(NatList) → NatList
Initial F-Algebras $F^*$ are minimal F-Algebras:

\[
\begin{aligned}
F(F^*) & \xrightarrow{\text{intro}} F^* \\
F(g) & \\
F(A) & \xrightarrow{f} A
\end{aligned}
\]

\[\exists! g\]
Iteration

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

\[
g 0 = g \text{ (intro inl)} = f \text{ inl}
\]
\[
g (S \ n) = g \text{ (intro (inr } n\text{))} = f \text{ (inr (g } n\text{))}
\]
\[
\begin{align*}
g \ 0 & = f \ \text{inl} \\
g \ (S \ n) & = f \ (\text{inr} \ (g \ n))
\end{align*}
\]

So with \( a_0 := f \ \text{inl} : A \) and \( f_0 := f \circ \text{inr} : A \rightarrow A \)

\[
\begin{align*}
g \ 0 & = a_0 \\
g \ (S \ n) & = f_0 \ (g \ n)
\end{align*}
\]

and therefore

\[
g \ n = f_0^n \ a_0
\]

On the other hand for every

\[
a_0 : A \quad f_0 : A \rightarrow A
\]

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

\[ a_0 : A \]
\[ f_0 : \mathbb{N} \to A \to A \]

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

\[ g(0) = a_0 \]
\[ g(Sn) = f_0 n (g n) \]

This allows to define efficiently the inverse of \( S \):

\[ \text{pred} : \mathbb{N} \to \mathbb{N} \]
\[ \text{pred } 0 = 0 \]
\[ \text{pred } (Sn) = n \]
Recursion

We have

\[
\begin{align*}
a_0 & : A \\
f_0 & : \mathbb{N} \rightarrow A \rightarrow A
\end{align*}
\]

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

\[\mathbb{N} \times A\]

Define

\[
\begin{align*}
f : (1 + (\mathbb{N} \times A)) & \rightarrow (\mathbb{N} \times A) \\
f \text{ inl} & = \langle 0, a_0 \rangle \\
f (\text{inr } \langle n, a \rangle) & = \langle S\ n, f_0\ n\ a \rangle
\end{align*}
\]
\[ f \text{ inl} = \langle 0, a \rangle \]
\[ f \left( \text{inr} \langle n, a \rangle \right) = \langle n + 1, f_0 n a \rangle \]

Both \( \pi_0 \circ g \) and \( \text{id} \) make the outermost diagram commute.

By uniqueness follows \( \pi_0 \circ g = \text{id} \),
therefore \( g \ n = \langle n, g_0 \ n \rangle \) for some \( g_0 : \mathbb{N} \to A \).
\[ f \text{ inl} = \langle 0, a \rangle \]
\[ f \text{ (inr } \langle n, a \rangle \rangle) = \langle n + 1, f_0 \ n \ a \rangle \]
\[ g \ n = \langle n, g_0 \ n \rangle \]

Therefore

\[ g_0 \ 0 = \pi_1(g \text{ (intro inl )}) = \pi_1(f \text{ inl }) = a_0 \]
\[ g_0 \ (S \ n) = \pi_1(g \text{ (intro (inr n))}) = \pi_1(f \text{ (inr } \langle n, g_0 \ n \rangle \rangle) = f_0 \ n \ (g_0 \ n) \]
Induction can be regarded as dependent elimination:

Assume

\[
\begin{align*}
A & : \mathbb{N} \rightarrow \text{Set} \\
a_0 & : A 0 \\
f_0 & : (n : \mathbb{N}) \rightarrow A n \rightarrow A (S n)
\end{align*}
\]

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

\[
\begin{align*}
g 0 & = a_0 \\
g (S n) & = f_0 n (g n)
\end{align*}
\]

Can be derived in the same way as recursion.
Initial Algebras in Functional Programming

Coalgebras and Copatterns

Codata Types

Conclusion
Final coalgebras $F^\infty$ are obtained by reversing the arrows in the diagram for $F$-algebras:

\[
\begin{array}{ccc}
A & \xrightarrow{f} & F(A) \\
& \swarrow_{\exists!g} & \searrow^{F(g)} \\
F^\infty & \xrightarrow{\text{case}} & F(F^\infty)
\end{array}
\]
Consider Streams = $F^\infty$ where $F(X) = \mathbb{N} \times X$:

\[
\begin{array}{c}
A \xrightarrow{f} \mathbb{N} \times A \\
\exists! g \\
\text{Stream} \xrightarrow{\text{case}} \mathbb{N} \times \text{Stream}
\end{array}
\]

Let

\[
\text{case } s = \langle \text{head } s, \text{tail } s \rangle
\]

and

\[
f \ a = \langle f_0 \ a, f_1 \ a \rangle
\]
Coalgebras and Copatterns

Guarded Recursion

\[ A \xrightarrow{\langle f_0, f_1 \rangle} \mathbb{N} \times A \]

\[ \exists! g \]

\[ \text{Stream} \xrightarrow{\langle \text{head}, \text{tail} \rangle} \mathbb{N} \times \text{Stream} \]

\[ \text{id} \times g \]

Resulting equations:

\[
\begin{align*}
\text{head} (g \ a) &= f_0 \ a \\
\text{tail} (g \ a) &= g (f_1 \ a)
\end{align*}
\]
Example of Guarded Recursion

\[
\begin{align*}
\text{head } (g \ a) &= f_0 \ a \\
\text{tail } (g \ a) &= g \ (f_1 \ a)
\end{align*}
\]

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:

\[
\begin{align*}
\text{inc} : \mathbb{N} &\to \text{Stream} \\
\text{head } (\text{inc } n) &= n \\
\text{tail } (\text{inc } n) &= \text{inc } (n + 1)
\end{align*}
\]
Coalgebras and Copatterns

Corecursion

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

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

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

$$\begin{align*}
A & : \text{Set} \\
 f_0 & : A \rightarrow \mathbb{N} \\
 f_1 & : A \rightarrow (\text{Stream } + \ A)
\end{align*}$$

we get $$g : A \rightarrow \text{Stream}$$ s.t.

$$\begin{align*}
\text{head } (g\ a) & = f_0\ a \\
\text{tail } (g\ a) & = s \quad \text{ if } f_1\ a = \text{inl } s \\
\text{tail } (g\ a) & = g\ a' \quad \text{ if } f_1\ a = \text{inr } a'
\end{align*}$$
Iteration and Recursion

(I learned this symmetry from Peter Hancock)

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

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

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

$$
\begin{align*}
f &: \mathbb{N} \to A \\
f 0 &= a_0 \\
f (S \ n) &= f_0 \langle n, f \ n \rangle
\end{align*}
$$
Coiteration and Corecursion

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

\[
    f : A \to \text{Stream} \\
    \text{head } (f \ a) = f_0 \ a \\
    \text{tail } (f \ a) = f \ (f_1 \ a)
\]

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

\[
    f : A \to \text{Stream} \\
    \text{head } (f \ a) = f_0 \ a \\
    \text{tail } (f \ a) = s \quad \text{if} \quad f_1 \ a = \text{inl} \ s \\
    \text{tail } (f \ a) = f \ a' \quad \text{if} \quad f_1 \ a = \text{inr} \ a'
\]
Recursion, Corecursion

Recursion allows to define the inverse of the constructor \( S \):

\[
\begin{align*}
pred : \mathbb{N} &\to \mathbb{N} \\
pred 0 &\ = \ 0 \\
pred (S \ n) &\ = \ n
\end{align*}
\]

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

\[
\begin{align*}
\text{cons} : \mathbb{N} &\to \text{Stream} \to \text{Stream} \\
\text{head} (\text{cons} \ n \ s) &\ = \ n \\
\text{tail} (\text{cons} \ n \ s) &\ = \ s
\end{align*}
\]
Nested Corecursion

\[
\text{stutter} : \mathbb{N} \to \text{Stream} \\
\text{head} (\text{stutter } n) = n \\
\text{head} (\text{tail} (\text{stutter } n)) = n \\
\text{tail} (\text{tail} (\text{stutter } n)) = \text{stutter} (n + 1)
\]

Even more general schemata can be defined.
Weakly Final Coalgebra

- Equality for final coalgebras is undecidable:
  Two streams
  \[ s = (a_0, a_1, a_2, \ldots) \]
  \[ t = (b_0, b_1, b_2, \ldots) \]
  are equal iff \( a_i = b_i \) for all \( i \).

- Even the weak assumption
  \[ \forall s. \exists n, s'. s = \text{cons } n \ s' \]
  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.
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

\[ f \ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1, \ldots \]

\[ f : \mathbb{N} \to \text{Stream} \]

\[ f = \ ? \]
Patterns and Copatterns

\[ f \ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1, \ldots \]

\[ f : \mathbb{N} \rightarrow \text{Stream} \]

Pattern match on \( f : \mathbb{N} \rightarrow \text{Stream} \):

\[ f : \mathbb{N} \rightarrow \text{Stream} \]

\[ f \ n = ? \]
Patterns and Copatterns

\[ f \ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1, \ldots \]

\[ f : \mathbb{N} \to \text{Stream} \]

\[ f \ n = ? \]

**Copattern matching** on \( f \ n : \text{Stream} \):

\[ f : \mathbb{N} \to \text{Stream} \]

\[ \text{head} \ (f \ n) = ? \]

\[ \text{tail} \ (f \ n) = ? \]
Patterns and Copatterns

\[ f \ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1, \ldots \]

\[ f : \mathbb{N} \rightarrow \text{Stream} \]
\[ \text{head} \ (f \ n) = ? \]
\[ \text{tail} \ (f \ n) = ? \]

**Pattern matching** on the first \( n : \mathbb{N} \):

\[ f : \mathbb{N} \rightarrow \text{Stream} \]
\[ \text{head} \ (f \ 0) = ? \]
\[ \text{head} \ (f \ (S \ n)) = ? \]
\[ \text{tail} \ (f \ n) = ? \]
Patterns and Copatterns

\[ f \ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1, \ldots \]

\[ f : \mathbb{N} \rightarrow \text{Stream} \]
\[ \text{head} \ (f \ 0) = ? \]
\[ \text{head} \ (f \ (S \ n)) = ? \]
\[ \text{tail} \ (f \ n) = ? \]

Pattern matching on second \( n : \mathbb{N} \):

\[ f : \mathbb{N} \rightarrow \text{Stream} \]
\[ \text{head} \ (f \ 0) = ? \]
\[ \text{head} \ (f \ (S \ n)) = ? \]
\[ \text{tail} \ (f \ 0) = ? \]
\[ \text{tail} \ (f \ (S \ n)) = ? \]
Patterns and Copatterns

\[ f \ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1, \ldots \]

\[ f : \mathbb{N} \to \text{Stream} \]

\[ \text{head} \ (f \ 0) = ? \]
\[ \text{head} \ (f \ (S \ n)) = ? \]
\[ \text{tail} \ (f \ 0) = ? \]
\[ \text{tail} \ (f \ (S \ n)) = ? \]

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

\[ f : \mathbb{N} \to \text{Stream} \]
\[ \text{head} \ (f \ 0) = ? \]
\[ \text{head} \ (f \ (S \ n)) = ? \]
\[ \text{head} \ (\text{tail} \ (f \ 0)) = ? \]
\[ \text{tail} \ (\text{tail} \ (f \ 0)) = ? \]
\[ \text{tail} \ (f \ (S \ n)) = ? \]
Coalgebras and Copatterns

Patterns and Copatterns

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

\[ f : \mathbb{N} \to \text{Stream} \]

\[ \text{head} \ (f\ 0) = \ ? \]

\[ \text{head} \ (f\ (S\ n)) = \ ? \]

\[ \text{head} \ \left(\text{tail} \ (f\ 0)\right) = \ ? \]

\[ \text{tail} \ \left(\text{tail} \ (f\ 0)\right) = \ ? \]

\[ \text{tail} \ \left(f\ (S\ n)\right) = \ ? \]

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

\[ f : \mathbb{N} \to \text{Stream} \]

\[ \text{head} \ (f\ 0) = \ ? \]

\[ \text{head} \ (f\ (S\ n)) = \ ? \]

\[ \text{head} \ \left(\text{tail} \ (f\ 0)\right) = \ ? \]

\[ \text{tail} \ \left(\text{tail} \ (f\ 0)\right) = \ ? \]

\[ \text{head} \ \left(\text{tail} \ (f\ (S\ n))\right) = \ ? \]

\[ \text{tail} \ \left(\text{tail} \ (f\ (S\ n))\right) = \ ? \]
We resolve the goals:

\[ f : \mathbb{N} \rightarrow \text{Stream} \]

\[
\begin{align*}
\text{head} (f \ 0) &= 0 \\
\text{head} (\text{tail} (f \ 0)) &= 0 \\
\text{tail} (\text{tail} (f \ 0)) &= f \ N \\
\text{head} (f (S \ n)) &= S \ n \\
\text{head} (\text{tail} (f (S \ n))) &= S \ n \\
\text{tail} (\text{tail} (f (S \ n))) &= f \ n
\end{align*}
\]
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.

\[ t : A, \quad t \rightarrow t' \implies t' : A \]
Initial Algebras in Functional Programming

Coalgebras and Copatterns

Codata Types

Conclusion
Codata Type

- **Idea of Codata Types:**

  ```
  codata Stream : Setwhere
  cons : ℕ → Stream → Stream
  ```

- **Theoretical problem:**
  Underlying assumption is
  
  \[ \forall s : Stream. \exists n, s'. s = \text{cons } n s' \]

  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:

```haskell
data _==_ {A : Set} (a : A) : A → Set where
  refl : a == a

codata Stream : Set where
  cons : ℕ → Stream → Stream

zeros : Stream
zeros = cons 0 zeros

force : Stream → Stream
force s = case s of (cons x y) → cons x y

lem1 : (s : Stream) → s == force(s))
lem1 s = case s of (cons x y) → refl

lem2 : zeros == cons 0 zeros
lem2 = lem1 zeros
lem2 → refl  but ¬(refl : zeros == cons 0 zeros)
```
Multiple Constructors in Algebras and Coalgebras

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

```haskell
data N : Set where
  0 : N
  S : N → N
```

corresponds to

```haskell
data N : Set where
  intro : (1 + N) → N
```
Multiple Constructors in Algebras and Coalgebras

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

\[
\text{coalg Stream : Set where}
\]
\[
\text{head : Stream } \to \mathbb{N}
\]
\[
\text{tail : Stream } \to \text{Stream}
\]

corresponds to

\[
\text{coalg Stream : Set where}
\]
\[
\text{case : Stream } \to (\mathbb{N} \times \text{Stream})
\]
Codata Types Correspond to Disjoint Union

Consider

\[
\begin{align*}
codata \ coList : \ Set \ where \\
nil & : \ coList \\
cons & : \ \mathbb{N} \to coList \to coList
\end{align*}
\]

Cannot be simulated by a coalgebra with several destructors.
Simulating Codata Types by Simultaneous Algebras/Coalgebras

- Represent Codata as follows

  ```
  mutual
  coalg coList : Set where
  unfold : coList → coListShape

  data coListShape : Set where
  nil : coListShape
  cons : ℕ → coList → coListShape
  ```
Definition of Append

\[
\text{append} : \text{coList} \rightarrow \text{coList} \rightarrow \text{coList}
\]
\[
\text{append } l \; l' = ?
\]
Definition of Append

append : coList → coList → coList
append ℓ ℓ′ =?

We copattern match on append ℓ ℓ′ : coList:

append : coList → coList → coList
unfold (append ℓ ℓ′) =?
Definition of Append

append : coList → coList → coList
unfold (append l l′) =?

We cannot pattern match on l.
But we can do so on (unfold l):

append : coList → coList → coList
unfold (append l l′) =
case (unfold l) of
  nil         → ?
  (cons n l)  → ?
Definition of Append

append : coList → coList → coList

unfold (append l l′) =
  case (unfold l) of
    nil        →  ?
    (cons n l) →  ?

We resolve the goals:

append : coList → coList → coList

unfold (append l l′) =
  case (unfold l) of
    nil        →  unfold l′
    (cons n l) →  cons n (append l l′)
Fibonacci Numbers

Efficient Haskell version adapted to our codata notation:

```haskell
codata Stream : Set where
    cons : ℕ → Stream → Stream

tail : Stream → Stream
    tail (cons n l) = l

addStream : Stream → Stream → Stream
    addStream (cons n l) (cons n' l') = cons (n + n') (addStream l l')

fib : Stream
    fib = cons 1 (cons 1 (addStream fib (tail fib)))
```

Requires lazy evaluation.
Fibonacci Numbers using Coalgebras

calg Stream : Set where
   head   : Stream → ℕ
   tail   : Stream → Stream

addStream : Stream → Stream → Stream
head (addStream l l') = head l + head l'
tail (addStream l l') = addStream (tail l) (tail l')

fib : Stream
head fib = 1
head (tail fib) = 1
tail (tail fib) = addStream fib (tail fib)

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

Initial Algebras in Functional Programming

Coalgebras and Copatterns

Codata Types

Conclusion
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.