Coalgebraic Programming Using Copattern Matching

Anton Setzer

Swansea University, Swansea UK

Gregynog, Wales, UK, 27 June 2013

Continuity, Computability, Constructivity – From Logic to Algorithms (CCC 2013)
Axiomatising the Real Numbers in Dependent Type Theory

Formulation of Coalgebras in Dependent Type Theory

Patterns and Copatterns

Conclusion

Appendix: Definition of Example of (Co)pattern Matching in Stages

Appendix: Simulating Codata Types in Coalgebras
Axiomatising the Real Numbers in Dependent Type Theory

Formulation of Coalgebras in Dependent Type Theory

Patterns and Copatterns

Conclusion

Appendix: Definition of Example of (Co)pattern Matching in Stages

Appendix: Simulating Codata Types in Coalgebras
We want to formulate real numbers in dependent type theory.
Instead of working with concrete computable real numbers we want to work with

- axiomatized abstract real numbers,
- and a predicate for real numbers being computable.

Then we show that functions we want to define map computable real numbers to computable ones.
From this we obtain an algorithm for computing the function on suitable representations.
Postulates

- The theorem prover Agda has the concept of a postulate.
- postulate \( a : A \)
  means that we introduce a new constant \( a \) of type \( A \) without any computation rules.
- As in any axiomatic approach, postulates can make Agda inconsistent:
  \[
  \text{postulate falsum} : \bot
  \]
  allows us to prove everything.
- Postulates are okay, if one allows them in a restricted way.
Real Number Axioms in Agda

postulate \( \mathbb{R} \) : Set
postulate zero : \( \mathbb{R} \)
postulate \( _+_- \) : \( \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \)
postulate ax+ : (\( r : \mathbb{R} \)) \rightarrow r + 0 == r 
...

Signed Digit Reals

Digit = \{-1, 0, 1\} : Set

codata SignedDigit : \mathbb{R} \rightarrow \text{Set} \text{ where}

\text{signedDigit} : (r : \mathbb{R})
\quad \rightarrow (r \in [-1, 1])
\quad \rightarrow (d : \text{Digit})
\quad \rightarrow \text{SignedDigit} (2 \ast r - d)
\quad \rightarrow \text{SignedDigit} r

We can extract from a proof of SignedDigit the nth Digit:

\text{signedDigit\_to\_nthDigit} : (r : \mathbb{R}) \rightarrow (\text{SignedDigit} r) \rightarrow \mathbb{N} \rightarrow \text{Digit}
We want that if we prove for some \( r \)

\[ p : \text{SignedDigit} \ r \]

then

\[ \text{signedDigit}_{\text{to}}_{\text{nthDigit}} \ r \ p \ 17 \]

reduces to \(-1\) or \(0\) or \(1\)
and not to something like

\[ \text{axiom1} \ (\text{axiom2} \ 5) \ 6 \]

For this we need to make sure that from a postulated axioms we cannot extract any computational content.

What we want is that if we derive

\[ a : A \]

where \( A \) algebraic data type, \( a \) is closed, then \( a \) is canonical, i.e. starts with a constructor.
Postulated functions have as result type equalities or postulated types.

- Especially negation is not allowed as conclusion because of $\neg (\neg A = A) \rightarrow \bot$.

Functions defined by case distinction on equalities have as result type only equalities or postulated types.

- So when using postulated functions and equalities we stay within equalities and postulated types.
Equalities

- The problem with equalities was that they occur in conclusions in Agda.
- If we had 2 equalities:
  - one on postulated types,
  - one on non-postulated types,
  then only a restriction on the equality on postulated types is needed.

Chi Ming Chuang
- showed that under these conditions all closed elements algebraic types are canonical,
- introduced the signed digit real numbers and showed that they are closed under $\text{av}$, $\ast$ and contain the rationals,
- transformed them into programs computing those operations on Reals given by streams of signed digits,
- was able to execute the resulting programs using a compiled version of Agda.
Axiomatising the Real Numbers in Dependent Type Theory

Formulation of Coalgebras in Dependent Type Theory

Patterns and Copatterns

Conclusion

Appendix: Definition of Example of (Co)pattern Matching in Stages

Appendix: Simulating Codata Types in Coalgebras
Codata in Functional Programming

- SignedDigit above was defined by a codata type.
- Consider a simpler example:

\[
\text{codata Stream : Set where} \\
\text{cons : } \mathbb{N} \rightarrow \text{Stream} \rightarrow \text{Stream}
\]

Codata contains objects such as

\[
\text{cons 0 (cons 0 (cons 0 \cdots))}
\]

- We immediately get non-normalisation.
- Restrictions were applied in Coq and Agda on reductions of elements of codata types.
  - In Coq resulted in problem of subject reduction.
  - In Agda restrictions make codata type not very useful.
Coalgebras

- Solution is to use approach from category theory.
- Treat coalgebras as we treat functions in the $\lambda$-calculus:
  - There functions are not a set of pairs
    - and therefore an infinite object,
  - but a program which applied to its arguments computes the result.
- Similarly elements of coalgebras are not per se infinite objects, but objects which can be unfolded computationally possibly infinitely often:

\[
\text{coalg } \text{Stream} : \text{Set where}
\]
\[
\text{head} : \text{Stream} \rightarrow \mathbb{N}
\]
\[
\text{tail} : \text{Stream} \rightarrow \text{Stream}
\]
- Idea is: an element of Stream is any object, to which we can apply head and tail and obtain natural numbers or Streams.
Introduction Rule

\[
\text{coalg Stream : Set where}
\]
\[
\begin{align*}
\text{head} & : \text{Stream} \rightarrow \mathbb{N} \\
\text{tail} & : \text{Stream} \rightarrow \text{Stream}
\end{align*}
\]

- Elimination rule for Stream is given by it’s eliminators head, tail.
- Introduction rule is “derived” (not in a mathematical sense) from the principle that elements of Stream are anything admitting head and tail.
- Example:

\[
\begin{align*}
\text{inc} : \mathbb{N} & \rightarrow \text{Stream} \\
\text{head} (\text{inc } n) & = n \\
\text{tail} (\text{inc } n) & = \text{inc } (n + 1)
\end{align*}
\]
Introduction Rules for Coalgebras

- In its simple form (coiteration) elimination rules correspond exactly to the categorical diagram of a weakly final coalgebra.
- More advanced forms (e.g. corecursion) can be derived for final coalgebras and then used to extend weakly final coalgebras.
Axiomatising the Real Numbers in Dependent Type Theory

Formulation of Coalgebras in Dependent Type Theory

Patterns and Copatterns

Conclusion

Appendix: Definition of Example of (Co)pattern Matching in Stages

Appendix: Simulating Codata Types in Coalgebras
Patterns and Copatterns

- In our POPL 2013 paper
  - Andreas Abel, Brigitte Pientka, David Thibodeau and Anton Setzer: Copatterns: programming infinite structures by observations. POPL 2013, pp. 27 - 38

  we
  - showed how to mix pattern and copattern matching, and nest them as well,
  - introduced a small (non-normalising) calculus for mixed and nested pattern and copattern matching,
  - showed that this guarantees that all function definitions are coverage complete,
  - showed that the resulting calculus fulfils subject reduction.
Example of Patterns and Copatterns

Definition of the stream:
\[ f \ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1, \ldots \]

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

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

- There is an easy algorithm to reduce these definitions back to case distinction operators and full recursion.
- One can trace back the recursion and in some cases reduce it to the primitive (co)recursion operators.
Let $I$ denote $[-1, 1]$.
Let $I^I$ be $I \rightarrow I$ for a postulated $I$ or postulated type with suitable axioms.

$$\text{coalg } \text{Cont } (f : I^I) : \text{Set where}$$
$$\text{elim} : (f : I^I) \rightarrow \text{Cont } f \rightarrow \text{Contaux } f$$

$$\text{data } \text{Contaux } (f : I^I) : \text{Set where}$$
$$\text{consume} : (f : I^I) \rightarrow ((d : \text{Digit}) \rightarrow \text{Contaux } (f \circ e_d)) \rightarrow \text{Contaux } f$$
$$\text{produce} : (f : I^I) \rightarrow (d : \text{Digit}) \rightarrow \text{Cont} (e_d^{-1} \circ f) \rightarrow \text{Contaux } f$$
Conclusion

Axiomatising the Real Numbers in Dependent Type Theory

Formulation of Coalgebras in Dependent Type Theory

Patterns and Copatterns

Conclusion

Appendix: Definition of Example of (Co)pattern Matching in Stages

Appendix: Simulating Codata Types in Coalgebras
Use of postulated Real numbers very good approach to treating real numbers in type theory.

Restrictions on postulates guarantee that program extraction works.

Copattern matching is the correct dual of pattern matching.

Definition of functions on data and codata types by pattern/copattern matching works well.
Axiomatising the Real Numbers in Dependent Type Theory

Formulation of Coalgebras in Dependent Type Theory

Patterns and Copatterns

Conclusion

Appendix: Definition of Example of (Co)pattern Matching in Stages

Appendix: Simulating Codata Types in Coalgebras
Patterns and Copatterns

- We demonstrate this by an example:
- Example define stream:
  \[ f \ n = \]
  \[ n, n, n - 1, n - 1, \ldots 0, 0, N, N, N - 1, N - 1, \ldots 0, 0, N, N, N - 1, N - 1, \ldots \]
Appendix: Definition of Example of (Co)pattern Matching in Stages

Patterns and Copatterns

\[ f \ n = n, n, n-1, n-1, \ldots 0, 0, N, N, N-1, N-1, \ldots 0, 0, N, N, N-1, N-1, \ldots \]
\[ f : \mathbb{N} \rightarrow \text{Stream} \]
\[ f = ? \]
Patterns and Copatterns

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

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

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

\[ f : \mathbb{N} \rightarrow \text{Stream} \]
\[ f \ n = ? \]
Patterns and Copatterns

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

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

\[ f \ n = ? \]

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

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

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

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

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

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

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

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

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

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

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

**Copattern matching**

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

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

We resolve the goals:

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

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

- There is an easy algorithm to reduce these definitions back to case distinction operators and full recursion.
- One can trace back the recursion and in some cases reduce it to the primitive (co)recursion operators.
Appendix: Simulating Codata Types in Coalgebras

Axiomatising the Real Numbers in Dependent Type Theory

Formulation of Coalgebras in Dependent Type Theory

Patterns and Copatterns

Conclusion

Appendix: Definition of Example of (Co)pattern Matching in Stages

Appendix: Simulating Codata Types in Coalgebras
Having more than one constructor in algebras correspond to disjoint union:

```haskell
data \( \mathbb{N} \) : Set where
  0 : \( \mathbb{N} \)
  S : \( \mathbb{N} \rightarrow \mathbb{N} \)
```
corresponds to

```haskell
data \( \mathbb{N} \) : Set where
  intro : (1 + \( \mathbb{N} \)) \rightarrow \mathbb{N}
```
Multiple Constructors in Algebras and Coalgebras

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

\[
\text{coalg Stream : Set where}
\begin{align*}
\text{head} & : \text{Stream} \rightarrow \mathbb{N} \\
\text{tail} & : \text{Stream} \rightarrow \text{Stream}
\end{align*}
\]

Corresponds to

\[
\text{coalg Stream : Set where}
\begin{align*}
\text{case} & : \text{Stream} \rightarrow (\mathbb{N} \times \text{Stream})
\end{align*}
\]
Consider

codata coList : Set where
  nil : coList
  cons : ℕ → coList → coList

Cannot be simulated by using several destructors.
Simulating Codata Types by Simultaneous Algebras/Coalgebras

- Represent Codata as follows

```plaintext
mutual
coalg coList : Set where
    unfold : coList → coListShape
data coListShape : Set where
    nil     : coListShape
    cons   : ℕ → coList → coListShape
```
Definition of Append

append : coList → coList → coList
append l / l' =?
Definition of Append

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

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

\[
\text{append} : \text{coList} \rightarrow \text{coList} \rightarrow \text{coList}
\]
\[
\text{unfold} \; (\text{append} \; l \; l') = ?
\]
Appendix: Simulating Codata Types in Coalgebras

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′)