Coalgbras in Dependent Type Theory

Anton Setzer
Swansea University
Swanesa, UK

September 8, 2010
1. Categorical View of Coalgebras

2. Codata

3. Nils Danielsson’s ∞

4. Suggested Solution

5. Model
Algebraic Data Types

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

```haskell
data NatList : Set where
  nil  : NatList
  cons : ℕ → NatList → NatList
```
Algebraic Data Types

Notation:

\[ \text{nil}' + \text{cons}'(\mathbb{N}, X) \]

stands for the labelled disjoint union, i.e. the set \( A \) s.t.

\[
\text{data } A : \text{Set where}
\]
\[
\text{nil}' : A
\]
\[
\text{cons}' : \mathbb{N} \rightarrow X \rightarrow A
\]

Let

\[
F : \text{Set} \rightarrow \text{Set}
\]
\[
F \; X = \text{nil}' + \text{cons}'(\mathbb{N}, X)
\]
Algebraic Data Types

\[ F \ X = \text{nil}' + \text{cons}'(\mathbb{N}, X) \]

Then the following is essentially equivalent to the definition of NatList:

\[
\text{data NatList : Set where}
\]

\[
\text{intro : } F \text{ NatList } \rightarrow \text{NatList}
\]

where

\[
\text{nil} = \text{intro \ nil'}
\]

\[
\text{cons \ n \ l} = \text{intro \ (cons' \ n \ l)}
\]
Categorical View of Initial Algebras

The introduction elimination and equality rules for algebraic data types follow then from the diagram for initial \( F \)-algebras (denoted by \( \mu F \))

\[
\begin{array}{ccc}
F (\mu F) & \xrightarrow{\text{intro}} & \mu F \\
F g & \downarrow & \exists! g \\
F A & \xrightarrow{f} & A
\end{array}
\]

One writes \( \mu X.t \) for \( \mu (\lambda X.t) \) e.g.

\[
\text{NatList} = \mu X.\text{nil'} + \text{cons'}(\mathbb{N}, X)
\]
Final Coalgebras $\nu F$ are obtained by reversing the arrows:

$$A \xrightarrow{f} FA$$

$$\exists! g \downarrow \quad \quad \quad \quad Fg$$

$$\nu F \xrightarrow{\text{case}} F(\nu F)$$

Again we write $\nu X.t$ for $\nu (\lambda X.t)$.

In weakly final coalgebras the uniqueness of $g$ is omitted.

Coalgebras can be used to model interactive programs and objects from object-oriented programming in dependent type theory.
1. Categorical View of Coalgebras

Suggested Notation

calg NatColist : Set where
case : NatColist → nil + cons(ℕ, NatColist)

▶ To an element of NatColist as above we can apply casedistinction as above.
▶ Furthermore from the finality we can derive the principle of guarded recursion:
   We can define \( f : A \to NatColist \) by saying what case \( (f \ a) \) is:
   ▶ nil
   ▶ cons \( n \ l \) for some \( n : \mathbb{N}, l : \text{NatColist} \)
   ▶ cons \( n \ (f \ a') \) for some \( n : \mathbb{N}, a : A \).
Example

inclist : \( \mathbb{N} \rightarrow \text{NatColist} \) where
\[
\text{case } (\text{inclist } n) = \text{cons } n \ (\text{inclist } (n + 1))
\]

Main goal of this talk: To define nice notations so that coalgebras become usable.
1. Categorical View of Coalgebras

2. Codata

3. Nils Danielsson’s ∞

4. Suggested Solution

5. Model
Coalgebras were introduced in programming languages as codata types:

```plaintext
codata NatColist : Set where
  nil      : NatColist
  cons     : ℕ → NatColist → NatColist
```

Idea is that elements of NatColist are

- \( \text{cons } n_1 (\text{cons } n_2 (\text{cons } n_3 \cdots (\text{cons } n_k \text{ nil}) \cdots )) \) or
- \( \text{cons } n_1 (\text{cons } n_2 (\text{cons } n_3 \cdots . \cdots ) \cdots ) \)
Problem of codata

- No normalisation, e.g.
  \[ \text{inclist } 0 = \text{cons } 0 (\text{cons } 1 (\text{cons } 2 \cdots )) \]

- Undecidability of equality.
  \[ \text{cons } (f \ 0) (\text{cons } (f \ 1) \cdots ) = \text{cons } (g \ 0) (\text{cons } (g \ 1) \cdots ) \iff \forall n. f \ n = g \ n \]

In case of coalgebras

- Elements of coalgebras are not expanded indefinitely. They are only expanded if \textit{case} is applied to them.

- In case of weakly final coalgebras equality of elements of the coalgebras is equality of the underlying algorithms.
2. Codata

Denotational Problems of Coalgebras

\[
\begin{align*}
\text{coalg } \text{NatColist} & : \text{Set} \text{ where} \\
\text{case} & : \text{NatColist} \rightarrow \text{nil'} + \text{cons'}(\mathbb{N}, \text{NatColist}) \\
\text{case } (\text{inclist } n) & = \text{cons'} n (\text{inclist } (n + 1))
\end{align*}
\]

is much more lengthy than

\[
\begin{align*}
\text{codata } \text{NatColist} & : \text{Set} \text{ where} \\
\text{nil} & : \text{NatColist} \\
\text{cons} & : \mathbb{N} \rightarrow \text{NatColist} \rightarrow \text{NatColist} \\
\text{inclist } n & = \text{cons } n (\text{inclist } (n + 1))
\end{align*}
\]
Pseudo-Constructors

If we have

\[
\text{coalg \ NatColist : Set where}
\]
\[
\text{case : NatColist } \rightarrow \text{ nil' + cons'}(\mathbb{N}, \text{NatColist})
\]

we can define by guarded recursion

\[
\text{nil : NatColist where}
\]
\[
\text{case nil = nil'}
\]

\[
\text{cons : } \mathbb{N} \rightarrow \text{NatColist} \rightarrow \text{NatColist where}
\]
\[
\text{case (cons } n \text{ l) = cons' } n \text{ l}
\]
Pseudo-Constructors

However we do not have

\[
\text{case } a = \text{cons}'n l \implies a = \text{cons } n l
\]

So elements of NatColist are not of the form nil or cons \( n l \).

But behave like nil or cons \( n l \).
∼-Notation

Let

\[ s \sim t \iff \text{case } s = \text{case } t \]

Then we have

\[ \text{case } s = \text{nil}' \iff s \sim \text{nil} \]
\[ \text{case } s = \text{cons}' n \ l \iff s \sim \text{cons } n \ l \]

So if \( s : \text{NatColist} \) then

\[ s \sim \text{nil} \lor s \sim \text{cons } n \ l \text{ for some } n, l \]
1. Categorical View of Coalgebras

2. Codata

3. Nils Danielsson’s ∞

4. Suggested Solution

5. Model
Nils Danielsson’s ∞

Nils Danielsson and Thorsten Altenkirch suggested to have the following:

\[ \infty : \text{Set} \to \text{Set} \]
\[ \flat : \{ A : \text{Set} \} \to A \to \infty A \]
\[ \natural : \{ A : \text{Set} \} \to \infty A \to A \]

∞ A denote coalgebraic arguments in a definition, and one defines NatColist as

\[
\text{data NatColist : Set where}
\]
\[\text{nil : NatColist}\]
\[\text{cons : } \mathbb{N} \to \infty \text{NatColist} \to \text{NatColist}\]
What is $\infty A$?

$\infty A$ cannot mean

$$\nu X.A$$

since $\nu X.A$ is as a (non-weakly) final coalgebra isomorphic to $A$: With $F X = A$ we get

$$X \xrightarrow{f} F X = A$$

$$\exists!g \quad F g = \text{id}$$

$$A \xrightarrow{id} F A = A$$
What is $\infty A$?

What is meant by it is, that if $A$ is defined as an algebraic data type, $\infty A$ is defined mutually coalgebraically:

```haskell
data NatColist : Set where
  nil     : NatColist
  cons : N → $\infty$NatColist → NatColist
```

stands for

```haskell
data NatColist : Set where
  nil     : NatColist
  cons : N → $\infty$NatColist → NatColist
```

```
calg $\infty$NatColist : Set where
  ✷ : $\infty$NatColist → NatColist
```
Order between \textit{data/codata}

\begin{verbatim}
data NatColist : Set where
  nil      : NatColist
  cons     : \mathbb{N} \to \infty NatColist \to NatColist

coalg \infty NatColist : Set where
  \nu : \infty NatColist \to NatColist
\end{verbatim}

But there are two interpretations of the above:

1. \begin{align*}
F(X, Y) &= \text{nil} + \text{cons}(\mathbb{N}, Y).
G(X, Y) &= X \\
F'(Y) &= \mu X. F(X, Y) = \mu X.\text{nil} + \text{cons}(\mathbb{N}, Y) \\
& \xRightarrow{\text{R}} \text{nil} + \text{cons}(\mathbb{N}, Y) \\
\infty \text{ NatColist} &= \nu Y. G(F'(Y), Y) = \nu Y. F'(Y) \\
& \xRightarrow{\text{R}} \nu Y.\text{nil} + \text{cons}(\mathbb{N}, Y) \\
\text{NatColist} &= F'(\infty \text{ NatColist}) \\
& = \text{nil} + \text{cons}(\mathbb{N}, \infty \text{ NatColist})
\end{align*}
Order between data/codata

data NatColist : Set where
  nil : NatColist
  cons : \mathbb{N} \to \infty \text{NatColist} \to \text{NatColist}

coalg \infty \text{NatColist} : \text{Set where}
  \text{NatColist} : \text{NatColist} \to \text{NatColist}

2.

\begin{align*}
  G(X, Y) &= X \\
  F(X, Y) &= \text{nil} + \text{cons}(\mathbb{N}, Y). \\
  G'(X) &= \nu Y. G(X, Y) = \nu Y. X \\
  \cong X \\
  \text{NatColist} &= \mu X. F(X, G'(X)) \cong \mu X. F(X, X) \\
  &= \mu X. \text{nil} + \text{cons}(\mathbb{N}, X) \\
  \infty \text{NatColist} &= G'(\text{NatColist}) \\
  \cong \text{NatColist}
\end{align*}
Order between data/codata

First solution gives the desired result.

Origin of problem:

- If we have two functors $F(X, Y)$, and $G(X, Y)$ and if we want to minimize $X$ and maximize $Y$ there are two solutions:
  - Minimize $X$ as a functor depending on $Y$.
    Then maximize $Y$.
  - Maximize $Y$ as a functor depending on $X$.
    Then minimize $X$.

- With mutual data types this problem didn’t occur since if we minimize both $X$ and $Y$, the order doesn’t matter.
4. Suggested Solution

1. Categorical View of Coalgebras

2. Codata

3. Nils Danielsson's ∞

4. Suggested Solution

5. Model
In general we want to be able to form arbitrary combinations of $\mu$ and $\nu$. Idea: minimize and maximize in the order of occurrence.
data $A$ : Set where
  \( \text{intro}_0 : F(A, B, C, D) \rightarrow A \)
codata $B$ : Set where
  \( \text{case}_0 : B \rightarrow G(A, B, C, D) \)
data $C$ : Set where
  \( \text{intro}_1 : H(A, B, C, D) \rightarrow C \)
codata $D$ : Set where
  \( \text{case}_1 : D \rightarrow K(A, B, C, D) \)

to be interpreted as:

\[
\begin{align*}
F_0(Y, Z, Z') &= \mu X. F(X, Y, Z, Z') & \text{A in terms of } Y, Z, Z' \\
G_1(Z, Z') &= \nu Y. G(F'(Y, Z, Z'), Y, Z, Z') & \text{B in terms of } Z, Z' \\
F_1(Z, Z') &= F_0(G_1(Z, Z'), Z, Z') & \text{A in terms of } Z, Z' \\
H_2(Z') &= \mu Z. H(F_1(Z, Z'), G_1(Z, Z'), Z, Z') & \text{C in terms of } Z' \\
G_2(Z') &= G_1(H_2(Z'), Z') & \text{B in terms of } Z' \\
F_2(Z') &= F_1(H_2(Z'), Z') & \text{A in terms of } Z' \\
D &= \nu Z'. K(F_2(Z'), G_2(Z'), H_2(Z'), Z') & \text{Final Value of } D \\
C &= H_2(D) & \text{Final Value of } C \\
B &= G_2(D) & \text{Final Value of } B \\
A &= F_2(D) & \text{Final Value of } A
\end{align*}
\]
Example: \textbf{NatColist}

\begin{verbatim}
  data NatColist : Set where
    nil  : NatColist
    cons : \mathbb{N} \rightarrow \infty \text{NatColist} \rightarrow \text{NatColist}
\end{verbatim}

stands for

\begin{verbatim}
  data NatColist : Set where
    nil  : NatColist
    cons : \mathbb{N} \rightarrow \infty \text{NatColist} \rightarrow \text{NatColist}
\end{verbatim}

\begin{verbatim}
  coalg \infty \text{NatColist} : Set where
    \| : \infty \text{NatColist} \rightarrow \text{NatColist}
\end{verbatim}
inclist

inclist : \( \mathbb{N} \rightarrow \infty \) NatColist
\[ \downharpoonright (\text{inclist } n) = \text{cons } n (\text{inclist } (n + 1)) \]
or
\[ \text{inclist } n \sim \downharpoonright (\text{cons } n (\text{inclist } (n + 1))) \]

With
\[ s \triangleright t \iff \triangleright s = t \]

we get
\[ \text{inclist } n \triangleright \text{cons } n (\text{inclist } (n + 1)) \]
1. Categorical View of Coalgebras

2. Codata

3. Nils Danielsson’s ∞

4. Suggested Solution

5. Model
Form a term model with reduction rules corresponding to the equalities stated.

E.g. inclist is a function symbol with equality rule

\[ \text{case } (\text{inclist } n) = \text{cons } n (\text{inclist } (n + 1)) \]

Interpretation of \( \mu X. F(X) \):

\[
\llbracket \mu X. F(X) \rrbracket = \bigcap \{ X \subseteq \text{Term} \mid \text{intro}[\llbracket F(X) \rrbracket] \subseteq X \}
\]

Interpretation of \( \nu X. F(X) \):

\[
\llbracket \nu X. F(X) \rrbracket = \bigcup \{ X \subseteq \text{Term} \mid \text{case}[X] \subseteq \llbracket F(X) \rrbracket \}
\]
Conclusion

- Design decisions should be done by referring to the notion of coalgebras.
- Introduction of $\sim$ was a good decision, since it flags which equalities hold.
  If one uses $\infty$ only, one might need $\triangleright$.
- If $A$ is a data type referring to $\infty A$, then $\infty A$ gets is meaning as a coalgebra defined implicitly mutually after the definition of $A$.
- Order of algebras coalgebras matters.