Coalgebras in Dependent Type Theory

Anton Setzer
Swansea University
Swanesa, UK

September 8, 2010

Algebraic Data Types

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

$$
\begin{array}{l}
\text{data NatList} : \text{Set where} \\
\quad \begin{array}{lcl}
\text{nil} & : & \text{NatList} \\
\text{cons} & : & \mathbb{N} \to \text{NatList} \to \text{NatList}
\end{array}
\end{array}
$$

Algebraic Data Types

Notation:

$$\mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, X)$$

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

$$\begin{array}{l} \mathrm{data}\ A : \mathrm{Set}\ \mathrm{where} \\ \quad \begin{array}{lcl} \mathrm{nil}' & : & A \\ \mathrm{cons}' & : & \mathbb{N} \to X \to A \end{array} \end{array}$$

Let

$$\begin{array}{l} F : \mathrm{Set} \to \mathrm{Set} \\ F\ X = \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, X) \end{array}$$

Algebraic Data Types

$F\ X = \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, X)$

Then the following is essentially equivalent to the definition of $\mathrm{NatList}$:

$$\begin{array}{l} \mathrm{data}\ \mathrm{NatList} : \mathrm{Set}\ \ \mathrm{where} \\ \quad \mathrm{intro} \quad : \quad F\ \mathrm{NatList} \to \mathrm{NatList} \end{array}$$

where

$$\begin{array}{l} \mathrm{nil} = \mathrm{intro}\ \mathrm{nil}' \\ \mathrm{cons}\ n\ l = \mathrm{intro}\ (\mathrm{cons}'\ n\ l) \end{array}$$

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 $\mu\,\mathrm{F}$)

$$\begin{tikzcd}
\mathrm{F}\;(\mu\;\mathrm{F}) \arrow[r, "\mathrm{intro}"] \arrow[d, "\mathrm{F}\; g"'] & \mu\;\mathrm{F} \arrow[d, "\exists!\; g"] \\
\mathrm{F}\; A \arrow[r, "f"] & A
\end{tikzcd}$$

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

$$\mathrm{NatList} = \mu X.\mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, X)$$

Final Coalgebras

Final Coalgebras $\nu\, \mathrm{F}$ are obtained by reversing the arrows:

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "\exists! g"'] & \mathrm{F}\ A \arrow[d, "\mathrm{F}\ g"] \\ \nu\ \mathrm{F} \arrow[r, "\mathrm{case}"] & \mathrm{F}\ (\nu\ \mathrm{F}) \end{tikzcd}$$

Again we write $\nu X.t$ for $\nu\ (\lambda X.t)$.
In weakly final coalgbras the uniqueness of g is omitted.

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

Suggested Notation

$$\begin{array}{l} \mathrm{coalg}\ \mathrm{NatColist} : \mathrm{Set}\ \mathrm{where} \\ \quad \mathrm{case} : \mathrm{NatColist} \to \mathrm{nil} + \mathrm{cons}(\mathbb{N}, \mathrm{NatColist}) \end{array}$$

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

Example

$$\begin{array}{l}\mathrm{inclist} : \mathbb{N} \to \mathrm{NatColist}\ \ \mathrm{where} \\ \mathrm{case}\ (\mathrm{inclist}\ n) = \mathrm{cons}\ n\ (\mathrm{inclist}\ (n+1))\end{array}$$

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

Codata

Coalgebras were introduced in programming languages as codata types:

$$
\begin{array}{l}
\mathrm{codata}\ \mathrm{NatColist} : \mathrm{Set}\ \ \mathrm{where} \\
\quad \begin{array}{lcl}
\mathrm{nil} & : & \mathrm{NatColist} \\
\mathrm{cons} & : & \mathbb{N} \to \mathrm{NatColist} \to \mathrm{NatColist}
\end{array}
\end{array}
$$

Idea is that elements of $\mathrm{NatColist}$ are

- $\mathrm{cons}\ n_1\ (\mathrm{cons}\ n_2\ (\mathrm{cons}\ n_3\ \cdots (\mathrm{cons}\ n_k\ \mathrm{nil}) \cdots))$ or
- $\mathrm{cons}\ n_1\ (\mathrm{cons}\ n_2\ (\mathrm{cons}\ n_3\ \cdots .$

Problem of codata

- No normalisation, e.g.

$$\mathrm{inclist}\ 0 = \mathrm{cons}\ 0\ (\mathrm{cons}\ 1\ (\mathrm{cons}\ 2 \cdots))$$

- Undecidability of equality.

$$\begin{aligned}&\mathrm{cons}\ (f\ 0)\ (\mathrm{cons}\ (f\ 1)\ \ \cdots) = \mathrm{cons}\ (g\ 0)\ (\mathrm{cons}\ (g\ 1)\ \ \cdots)\\ &\Leftrightarrow \forall n.f\ \ n = g\ \ n\end{aligned}$$

In case of coalgebras

- Elements of coalgebras are not expanded indefinitely. They are only expanded if case is applied to them.
- In case of weakly final coalgebras equality of elements of the coalgebras is equality of the underlying algorithms.

Denotational Problems of Coalgebras

$$\begin{array}{l}\mathrm{coalg}\ \mathrm{NatColist} : \mathrm{Set}\ \ \mathrm{where} \\ \quad \mathrm{case} : \mathrm{NatColist} \to \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, \mathrm{NatColist})\end{array}$$

$$\mathrm{case}\ (\mathrm{inclist}\ n) = \mathrm{cons}'\ n\ (\mathrm{inclist}\ (n+1))$$

is much more lenghty than

$$\begin{array}{lcl}\mathrm{codata}\ \mathrm{NatColist} : \mathrm{Set}\ \ \mathrm{where} & & \\ \quad \mathrm{nil} & : & \mathrm{NatColist} \\ \quad \mathrm{cons} & : & \mathbb{N} \to \mathrm{NatColist} \to \mathrm{NatColist}\end{array}$$

$$\mathrm{inclist}\ n = \mathrm{cons}\ n\ (\mathrm{inclist}\ (n+1))$$

Pseudo-Constructors

If we have

$$\begin{array}{l} \text{coalg NatColist} : \text{Set where} \\ \quad \text{case} : \text{NatColist} \to \text{nil}' + \text{cons}'(\mathbb{N}, \text{NatColist}) \end{array}$$

we can define by guarded recursion

$$\begin{array}{l} \text{nil} : \text{NatColist where} \\ \text{case nil} = \text{nil}' \end{array}$$

$$\begin{array}{l} \text{cons} : \mathbb{N} \to \text{NatColist} \to \text{NatColist where} \\ \text{case } (\text{cons } n\ l) = \text{cons}'\ n\ l \end{array}$$

Pseudo-Constructors

However we do not have

$$\mathrm{case}\ a = \mathrm{cons}'\ n\ l \ \text{ implies } \ a = \mathrm{cons}\ n\ l$$

So elements of $\mathrm{NatColist}$ are not of the form nil or $\mathrm{cons}\ n\ l$.

But behave like nil or $\mathrm{cons}\ n\ l$.

$\sim$-Notation

Let

$$s \sim t \Leftrightarrow \mathrm{case}\ s = \mathrm{case}\ t$$

Then we have

$$\begin{aligned} \mathrm{case}\ s = \mathrm{nil}' &\Leftrightarrow s \sim \mathrm{nil} \\ \mathrm{case}\ s = \mathrm{cons}'\ n\ l &\Leftrightarrow s \sim \mathrm{cons}\ n\ l \end{aligned}$$

So if $s : \mathrm{NatColist}$ then

$$s \sim \mathrm{nil} \vee s \sim \mathrm{cons}\ n\ l \text{ for some } n, l$$

Nils Danielsson's ∞

Nils Danielsson and Thorsten Altenkirch suggested to have the following

$$\begin{array}{lcl} \infty & : & \mathrm{Set} \to \mathrm{Set} \\ \flat & : & \{A : \mathrm{Set}\} \to A \to \infty\ A \\ \natural & : & \{A : \mathrm{Set}\} \to \infty\ A \to A \end{array}$$

$\infty\ A$ denote coalgebraic arguments in a definition, and one defines $\mathrm{NatColist}$ as

$$\begin{array}{l} \mathrm{data\ NatColist} : \mathrm{Set\ where} \\ \quad \begin{array}{lcl} \mathrm{nil} & : & \mathrm{NatColist} \\ \mathrm{cons} & : & \mathbb{N} \to \infty\ \mathrm{NatColist} \to \mathrm{NatColist} \end{array} \end{array}$$

What is ∞ A?

∞ A cannot mean

$$\nu X.A$$

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

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "\exists ! g"'] & \mathrm{F}\ X = A \arrow[d, "\mathrm{F}\ g = \mathrm{id}"] \\
A \arrow[r, "\mathrm{id}"] & \mathrm{F}\ A = A
\end{tikzcd}$$

What is ∞ A?

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

$$
\begin{array}{l}
\text{data NatColist} : \text{Set where} \\
\quad \begin{array}{lcl}
\text{nil} & : & \text{NatColist} \\
\text{cons} & : & \mathbb{N} \to \infty\ \text{NatColist} \to \text{NatColist}
\end{array}
\end{array}
$$

stands for

$$
\begin{array}{l}
\text{data NatColist} : \text{Set where} \\
\quad \begin{array}{lcl}
\text{nil} & : & \text{NatColist} \\
\text{cons} & : & \mathbb{N} \to \infty\ \text{NatColist} \to \text{NatColist}
\end{array}
\end{array}
$$

$$
\begin{array}{l}
\text{coalg } \infty\ \text{NatColist} : \text{Set where} \\
\quad \natural : \infty\ \text{NatColist} \to \text{NatColist}
\end{array}
$$

Order between data/codata

$$
\begin{array}{l}
\text{data NatColist} : \text{Set where} \\
\quad \begin{array}{lcl} \text{nil} & : & \text{NatColist} \\ \text{cons} & : & \mathbb{N} \rightarrow \infty\ \text{NatColist} \rightarrow \text{NatColist} \end{array} \\
\text{coalg } \infty\ \text{NatColist} : \text{Set where} \\
\quad \natural : \infty\ \text{NatColist} \rightarrow \text{NatColist}
\end{array}
$$

But there are two interpretations of the above:

1.

$$
\begin{array}{lcl}
F(X, Y) & = & \text{nil} + \text{cons}(\mathbb{N}, Y). \\
G(X, Y) & = & X \\
F'(Y) & = & \mu X.F(X, Y) = \mu X.\text{nil} + \text{cons}(\mathbb{N}, Y) \\
 & \cong & \text{nil} + \text{cons}(\mathbb{N}, Y) \\
\infty\ \text{NatColist} & = & \nu Y.G(F'(Y), Y) = \nu Y.F'(Y) \\
 & \cong & \nu Y.\text{nil} + \text{cons}(\mathbb{N}, Y) \\
\text{NatColist} & = & F'(\infty\ \text{NatColist}) \\
 & = & \text{nil} + \text{cons}(\mathbb{N}, \infty\ \text{NatColist})
\end{array}
$$

Order between data/codata

$$
\begin{array}{l}
\text{data NatColist : Set where} \\
\quad \text{nil} \quad : \quad \text{NatColist} \\
\quad \text{cons} \quad : \quad \mathbb{N} \to \infty\,\text{NatColist} \to \text{NatColist} \\
\text{coalg } \infty\,\text{NatColist : Set where} \\
\quad \natural : \infty\,\text{NatColist} \to \text{NatColist}
\end{array}
$$

2.

$$
\begin{array}{lcl}
G(X,Y) & = & X \\
F(X,Y) & = & \text{nil} + \text{cons}(\mathbb{N}, Y). \\
G'(X) & = & \nu Y.G(X,Y) = \nu Y.X \\
 & \cong & X \\
\text{NatColist} & = & \mu X.F(X, G'(X)) \cong \mu X.F(X,X) \\
 & = & \mu X.\text{nil} + \text{cons}(\mathbb{N}, X) \\
\infty\,\text{NatColist} & = & G'(\text{NatColist}) \\
 & \cong & \text{NatColist}
\end{array}
$$

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.

Generality

In general we want to be able to form arbitrary combinations of μ and ν.
Idea: minimize and maximize in the order of occurrence.

$$
\begin{aligned}
&\text{data } A : \text{Set where} \\
&\quad \text{intro}_0 : F(A, B, C, D) \to A \\
&\text{codata } B : \text{Set where} \\
&\quad \text{case}_0 : B \to G(A, B, C, D) \\
&\text{data } C : \text{Set where} \\
&\quad \text{intro}_1 : H(A, B, C, D) \to C \\
&\text{codata } D : \text{Set where} \\
&\quad \text{case}_1 : D \to K(A, B, C, D)
\end{aligned}
$$

to be interpreted as:

$$
\begin{array}{lcll}
F_0(Y, Z, Z') & = & \mu X.F(X, Y, Z, Z') & A \text{ in terms of } Y, Z, Z' \\
G_1(Z, Z') & = & \nu Y.G(F'(Y, Z, Z'), Y, Z, Z') & B \text{ in terms of } Z, Z' \\
F_1(Z, Z') & = & F_0(G_1(Z, Z'), Z, Z') & A \text{ in terms of } Z, Z' \\
H_2(Z') & = & \mu Z.H(F_1(Z, Z'), G_1(Z, Z'), Z, Z') & C \text{ in terms of } Z' \\
G_2(Z') & = & G_1(H_2(Z'), Z') & B \text{ in terms of } Z' \\
F_2(Z') & = & F_1(H_2(Z'), Z') & A \text{ in terms of } Z' \\
D & = & \nu Z'.K(F_2(Z'), G_2(Z'), H_2(Z'), Z') & \text{Final Value of } D \\
C & = & H_2(D) & \text{Final Value of } C \\
B & = & G_2(D) & \text{Final Value of } B \\
A & = & F_2(D) & \text{Final Value of } A
\end{array}
$$

Example: NatColist

$$
\begin{array}{l}
\mathrm{data}\ \mathrm{NatColist} : \mathrm{Set}\ \mathrm{where} \\
\quad \begin{array}{lcl}
\mathrm{nil} & : & \mathrm{NatColist} \\
\mathrm{cons} & : & \mathbb{N} \to \infty\ \mathrm{NatColist} \to \mathrm{NatColist}
\end{array}
\end{array}
$$

stands for

$$
\begin{array}{l}
\mathrm{data}\ \mathrm{NatColist} : \mathrm{Set}\ \mathrm{where} \\
\quad \begin{array}{lcl}
\mathrm{nil} & : & \mathrm{NatColist} \\
\mathrm{cons} & : & \mathbb{N} \to \infty\ \mathrm{NatColist} \to \mathrm{NatColist}
\end{array}
\end{array}
$$

$$
\begin{array}{l}
\mathrm{coalg}\ \infty\ \mathrm{NatColist} : \mathrm{Set}\ \mathrm{where} \\
\quad \natural : \infty\ \mathrm{NatColist} \to \mathrm{NatColist}
\end{array}
$$

inclist

$$\begin{aligned}&\text{inclist} : \mathbb{N} \to \infty\ \text{NatColist}\\&\natural\ (\text{inclist}\ n) = \text{cons}\ n\ (\text{inclist}\ (n+1))\\&\text{or}\\&\text{inclist}\ n \sim \flat\ (\text{cons}\ n\ (\text{inclist}\ (n+1)))\end{aligned}$$

With

$$s \rhd t :\Leftrightarrow \natural s = t$$

we get

$$\text{inclist}\ n \rhd \text{cons}\ n\ (\text{inclist}\ (n+1))$$

Model

Form a term model with reduction rules corresponding to the equalities stated.
E.g. $\mathrm{inclist}$ is a function symbol with equality rule

$$\mathrm{case}\ (\mathrm{inclist}\ n) = \mathrm{cons}\ n\ (\mathrm{inclist}\ (n+1))$$

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

$$[\![\, \mu X.F(X) \,]\!] = \bigcap \{X \subseteq \mathrm{Term} \mid \mathrm{intro}[[\![\, F(X) \,]\!]] \subseteq X\}$$

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

$$[\![\, \nu X.F(X) \,]\!] = \bigcup \{X \subseteq \mathrm{Term} \mid \mathrm{case}[X] \subseteq [\![\, F(X) \,]\!]\}$$

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 ∞ 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.