Inductive types in constructive languages

Bruin, Peter Johan de

Chapter 7

Co-inductive types

We have noted in section 4.3 and 4.4 that the categories $\mathbf{TYPE}^N$ have initial F-algebras and initial $(F;E)$-algebras. Quite remarkably, the same holds for the opposite categories $(\mathbf{TYPE}^N)^{\mathsf{op}}$. Initial algebras in the opposite category are final co-algebras in the original category, and may be called *co-inductive types*. While the elements of initial F-algebras are like trees with finite branches only, elements of final F-coalgebras are like trees with possibly infinitely deep branches. Final coalgebras are introduced in 7.1.

In 7.2 we have a look at the various shapes which the unique homomorphism to a final coalgebra may take. We present the interesting example of infinite processes.

Section 7.3 shows how all recursion constructs that do not involve dependent functions dualize.

While adding equations to an initial F-algebra has the effect of identifying some trees (elements of the F-algebra), we prove in 7.4 that adding equations to a final F-coalgebra has the effect of removing some trees from the algebra. Section 7.5 contrasts this with the Algebraic Specification idea of final or terminal interpretation of equations over an initial algebra.

7.1 Dualizing F-algebras

Let $\mathcal{C}$ be the category $\mathbf{TYPE}^N$ for some type N. An F-algebra in $\mathcal{C}^{\mathsf{op}}$, say $(X{:}\,\mathcal{C}^{\mathsf{op}};\ \phi{:}\,F.X \to X \ \underline{\mathsf{in}}\ \mathcal{C}^{\mathsf{op}})$, is, by definition of $^{\mathsf{op}}$, a (Id, F)-algebra $(X{:}\,\mathcal{C};\ \phi{:}\,X \to F.X \ \underline{\mathsf{in}}\ \mathcal{C})$, which is also called an *F-coalgebra*. In section 8.2 we shall see that, if F is polynomial, then there exists a final F-coalgebra, which we name νF. Thus, for any F-coalgebra $(X;\phi)$ there is a unique homomorphism $f{:}\,(X;\phi) \to (U;\delta)$, with characteristic equation:

$$\delta \circ f \ = \ F.f \circ \phi \ . \tag{7.1}$$

This homomorphism is noted '$[\![X;\phi]\!]$', and called an *anamorphisms*, as devised by Erik Meijer. Note that δ is an isomorphism by theorem 4.2, as $(U;\delta)$ is initial in $\mathcal{C}^{\mathsf{op}}$.

Theorem 7.1 *A final F-coalgebra $(U;\delta)$ contains an initial F-algebra $(V;\delta^{\cup})$.*

Proof. Take $V{:}\,\mathcal{P}U := \bigcap(X \mid{:}\ \delta^{\cup} \in F.X \to X)$. Then $(V;\delta^{\cup})$ is initial by theorem 4.3 (no junk and no confusion). ∎

Whereas, in **TYPE**, elements of initial algebras are thought of as well-founded trees, final coalgebras do also contain all non-wellfounded trees, having infinitely deep branches.

Example 7.1 (Infinite lists) The algebra of streams or infinite lists $(E\infty; \langle \mathsf{hd}, \mathsf{tl}\rangle)$, as specified in example 3.8, is the final $(E\times)$-coalgebra.

To define the list $e\colon \mathbb{N}\infty$ of all even numbers, we let $f.n$ be the arithmetic sequence $\langle n, n+2, \ldots\rangle$, using an equation of the shape (7.1).

$$\langle \mathsf{hd}, \mathsf{tl}\rangle.(f.n) \;=\; (n, f.(n+2)) \;=\; (\,(\mathsf{I}_{\mathbb{N}} \times f) \circ \langle \mathsf{I}, (+2)\rangle\,).n$$

So $f := [\![\mathbb{N}; \langle \mathsf{I}, (+2)\rangle]\!]$. Then we take $e := f.0$.

Let us now define a bijection $g\colon E^\omega \leftrightarrow E\infty$. We wish

$$\begin{array}{rcl} g.e & = & f.0 \;\underline{\mathsf{where}} \\ & & \delta.(f.n) = (e_n, f.(n+1)) \end{array}$$

where δ is $\langle \mathsf{hd}, \mathsf{tl}\rangle$, so we define

$$g.e \;:=\; [\![\mathbb{N}; (n \mapsto (e_n, n+1))]\!].0 \;.$$

For the inverse, we define

$$g^\cup.l \;:=\; (n :: \mathsf{hd}.(\mathsf{tl}^{(n)}.l)) \;.$$

We prove $g \circ g^\cup = \mathsf{I}$:

$$\begin{array}{lcll} & g.(g^\cup.l) = l & & \\ \Leftrightarrow & f.0 = l \;\underline{\mathsf{where}}\; f := [\![\mathbb{N}; (n \mapsto ((g^\cup.l)_n, n+1))]\!] & & \{\text{definition } g\} \\ \Leftarrow & f = (n \mapsto \mathsf{tl}^{(n)}.l) & & \{\text{generalization}\} \\ \Leftrightarrow & \forall n :: \delta.(\mathsf{tl}^{(n)}.l) = (\mathsf{I} \times (n \mapsto \mathsf{tl}^{(n)}.l)).((g^\cup.l)_n, n+1) & & \{\text{anamorphism}\} \\ \Leftrightarrow & \forall n :: \mathsf{hd}.(\mathsf{tl}^{(n)}.l) = (g^\cup.l)_n \;\wedge\; \mathsf{tl}.(\mathsf{tl}^{(n)}.l) = \mathsf{tl}^{(n+1)}.l & & \{\text{pairing}\} \\ \Leftrightarrow & \mathsf{True} & & \end{array}$$

And $g^\cup \circ g = \mathsf{I}$:

$$\begin{array}{lcll} & g^\cup.(g.e) = e & & \\ \Leftrightarrow & \forall n :: \mathsf{hd}.(\mathsf{tl}^{(n)}.(f.0)) = e_n \quad \underline{\mathsf{where}}\; f := [\![\mathbb{N}; (n \mapsto (e_n, n+1))]\!] & & \\ \Leftarrow & \forall n, m :: \mathsf{hd}.(\mathsf{tl}^{(n)}.(f.m)) = e_{n+m} & & \\ \Leftarrow & \forall (m :: \mathsf{hd}.(\mathsf{tl}^{(0)}.(f.m)) = e_{m+0}\,) & & \\ & \wedge\, \forall (n, m :: \mathsf{hd}.(\mathsf{tl}^{(n+1)}.(f.m)) = \mathsf{hd}.(\mathsf{tl}^{(n)}.(f.(m+1))\,) & & \{\text{induction on } n\} \\ \Leftrightarrow & \mathsf{True} & & \{\text{definition } f\} \end{array}$$

7.1.1 Final F-algebras. Looking at final F-algebras in any category with final objects is not very interesting, for these are always that object (for $\mathbf{TYPE}^N$, the trivial unit algebra, all carriers having exactly one element). Dually, initial F-coalgebras are the initial object (for $\mathbf{TYPE}^N$, the empty algebra).

7.1.2 Dualizing plain algebras. In subsection 5.2.2 we gave an alternative scheme for mutually inductive types, plain algebras. This scheme seems less suited for dualization.

The operations were typed by:

$$\tau_{j:M}\colon \Pi(k\colon B_j :: T_{(djk)}) \to T_{(cj)} \ .$$

A dual algebra $(U;\delta)$ would be typed by

$$\delta_{j:M}\colon T_{(cj)} \to \Sigma(k\colon B_j :: T_{(djk)}) \ ,$$

for Σ is the generalized product constructor in the category $\mathbf{TYPE}^{\mathsf{op}}$. This is no longer a plain algebra. It seems not to be very useful because neither operations δ nor $\delta^{\cup}$ admit multiple arguments, and each operation having alternative single result types that are unrelated seems difficult to make sense of.

7.2 Anamorphism schemes

Equation (7.1) for an anamorphism $f\colon (X;\phi) \to (U;\delta)$ can take on a somewhat different shape if it is combined with pattern matching, and furthermore if F happens to be the sum of other functors.

First we eliminate δ on the left-hand side:

$$f \ = \ \delta^{\cup} \circ F.f \circ \phi \ . \tag{7.2}$$

Now, suppose that X can be split up over n patterns $\xi_i(x'\colon X'_i)\colon X$, so that:

$$\forall x\colon X :: \exists!i\colon < n;\ x'\colon X'_i :: x = \xi_i x' \ .$$

Then (7.2) may be written as a list of equations

$$f \circ \lambda\xi_i \ = \ \delta^{\cup} \circ F.f \circ \phi'_i \tag{7.3}$$

where $\phi'_i := \phi \circ \lambda\xi_i$.

Next, if F is a finite sum, $F.Y = \Sigma(j\colon < m :: F'_j.Y)$, one has actually a list of constructors

$$\tau_j\colon F'_j.U \to U \ := \ \delta^{\cup} \circ \sigma_j$$

or equivalently $[\tau] := \delta^{\cup}$. Then, if some ϕ'_i has the shape $\sigma_j \circ \phi''_i$, the respective equation (7.3) reduces to

$$f \circ \lambda\xi_i \ = \ \tau_j \circ F'_j.f \circ \phi''_i \ . \tag{7.4}$$

Example 7.2 (Processes) Co-inductive types offer elegant models for (indefinitely proceeding) processes, viewing these as incremental stream transformers. For example, the type of simple processes that can input data values from channels $i\colon I$ and output

data values over channels $o\colon O$, can make finitary nondeterministic choices and silent steps, and can halt, is the final coalgebra $(\mathsf{Sproc}(I,O);\delta)$ with $\mathsf{Sproc}(I,O)\colon \mathbf{Type}$ and

$$\begin{array}{rcl} \delta\colon \mathsf{Sproc}(I,O) \to & (& I \times (\mathsf{Data} \triangleright \mathsf{Sproc}(I,O)) \\ & + & O \times \mathsf{Data} \times \mathsf{Sproc}(I,O) \\ & + & \Sigma(n\colon \mathbb{N} :: \mathsf{Sproc}(I,O)^n) \\ & + & \mathsf{Sproc}(I,O) \\ & + & 1 \\ &) & \\ [\mathsf{outp},\mathsf{inp},\mathsf{choose},\mathsf{step},\mathsf{halt}] & := & \delta^{\cup} \end{array}$$

Here, $\mathsf{inp}.(i,s)$ represents a process that requires an input from channel i and continues with process si; $\mathsf{outp}.(o,c,s)$ outputs value c over channel o and continues with s; $\mathsf{choose}.(n;s)$ chooses some arbitrary $k\colon < n$ and continues with sk, and $\mathsf{halt}.0$ represents the process that just halts. $\mathsf{step}.s$ represents a process that performs some internal steps without external action, and continues with s.

The choose alternative may be omitted if one represents a nondeterministic process by a set of deterministic processes.

Now, consider the following program, written in CSP notation (Communicating Sequential Processes, Hoare [41]). It reads a number x from input channel A, and then repeats x times reading a number y from A and outputting y^2 to channel B.

$$A?x;\ *|[\ x > 0 \longrightarrow A?y;\ B!(y^2);\ x := x - 1\ ]| \ .$$

To give the process defined by this program, we first define X to be a suitable state space. Between each input or output action there has to be a distinguished state.

$$X\colon \mathbf{Type} \ := \ 1 + \mathbb{N} + \mathbb{N}^2$$

The mapping of states to processes, $f\colon X \to \mathsf{Sproc}(I,O)$, is then given by the following equations of the form (7.4).

$$\begin{array}{rcl} f.(0;0) & = & \mathsf{inp}.(A,(x :: f.(1;x))) \\ f.(1;x) & = & \underline{\mathsf{if}}\ x > 0\ \underline{\mathsf{then}}\ \mathsf{inp}.(A,(y :: f.(2;x,y)))\ \underline{\mathsf{else}}\ \mathsf{halt}.0 \\ f.(2;x,y) & = & \mathsf{outp}.(B,y^2,f.(1;x-1)) \end{array}$$

The intended process is now $f.(0;0)$, as the initial state is $(0;0)\colon X$.

Processes that operate in an environment that may be changed by the process itself can be modeled by final coalgebras in the category $\mathbf{TYPE}^N$, where type N is the set of possible environment states. This makes it possible to let the range of possible actions depend on the current environment state. (End of example)

Within a domain theory of partial and infinite objects (section 9.1), and hence in programming languages with partial objects, processes can be represented as continuous functions from lazy streams to lazy streams, which are also called *stream transformers*. Dybjer and Sander [25] represented a system of concurrent processes using the stream

approach. One needs a "network transfer function" to combine the separate agents into a single stream transformer. They used a functional calculus in which types are basically predicates so that final coalgebras can be obtained as the greatest fixed points of monotonic predicate transformers (as in section 10.2).

We refer to Malcolm [52] for some more examples and properties of final coalgebras, called "terminal data structures" there.

7.3 Dual recursion

We have a look at how the derived non-dependent recursors of chapter 6 dualize. Dependent recursors are not dualizable, as the dual of a dependent function would have to be something where the type of the argument depends on the function result, which is impossible.

All the following dual recursors can be transformed into a pattern-matching scheme like 7.4.

7.3.1 Algebraic recursion. As the scheme of algebraic recursion in section 6.1 works for any category with products, and products in category $\mathcal{C}^{\mathsf{op}}$ are sums in $\mathcal{C}$, we have the following corollary.

Corollary 7.2 *If $(T;\delta)$ is a final F-coalgebra in a category $\mathcal{C}$ with binary sums, then*

$$\frac{\begin{array}{l} U\colon\mathcal{C} \\ \psi\colon U \to F.(T+U) \end{array}}{\exists!\, f\colon U \to T :: \delta \circ f = F.[\mathsf{Id}, f] \circ \psi} \tag{7.5}$$

Example 7.3 Let $(T;\delta) := \nu(X \mapsto E \times X^2)$ be the coalgebra of non-wellfounded labeled binary trees. Given a label $e\colon E$ and a tree $t\colon T$, we can construct a tree s with

$$\delta.s = (e, s, t)$$

by taking $s := f.0$ where $f\colon 1 \to T$ is, using (7.5), the unique solution to:

$$\delta \circ f = (\mathsf{I}_E \times [\mathsf{I}_T, f]^2) \circ \mathsf{K}(e, ((1;0),(0;t))) \ .$$

So we apply (7.5) to $\psi := \mathsf{K}(e, ((1;0),(0;t)))$.

7.3.2 Mendler recursion. Mendler's recursor dualizes too (only for non-dependent functions of course) to:

$$\frac{\begin{array}{l} U\colon \mathbf{Type}; \\ X\colon \mathbf{Type};\ i\colon T \to X;\ h\colon U \to X \vdash s_X(i,h)\colon U \to F.X \\ \text{where } s \text{ is } \textit{natural} \end{array}}{\exists!\, f\colon U \to T :: \delta \circ f = s_T(\mathsf{I}, f)} \tag{7.6}$$

Example 7.4 The same $s\colon T$ as in example 7.3 is obtained by taking for $f\colon 1 \to T$ the unique solution to:

$$\delta \circ f = \mathsf{K}(e, f.0, \mathsf{I}.t) \ .$$

So we apply (7.6) to $s_X(i,h) := \mathsf{K}(e, h.0, i.t)$. Check that this is well-typed.

7.3.3 Liberal mutual recursion. Rule (6.11) dualizes trivially, yielding an $f{:}\,U \to T_d$ in $\mathbf{TYPE}^M$ that satisfies $\delta_d \circ f = F_d.[\mathsf{Id}, f_=] \circ \psi$.

7.4 Dual equations

We shall now look at the effect of adding equations to a final coalgebra, by applying the categorical notion of an algebra with equations, as described in paragraph 4.4.3, to the dual category $\mathcal{C}^{\mathsf{op}}$ where $\mathcal{C} := \mathbf{TYPE}^N$. This is not to be confused with the terminal interpretation of equations in algebraic specification, described in section 7.5.

A law E in the category $\mathcal{C}^{\mathsf{op}}$ is a functor $H{:}\,\mathcal{C} \to \mathcal{C}$ with two natural transformations $E_j{:}\,U \stackrel{\cdot}{\to} HU$, where U is the forgetful functor $(X;\phi) \mapsto X$. An F-coalgebra $(X;\phi)$ satisfies law E when $E_0(X;\phi) =_{X \to H.X} E_1(X;\phi)$, that is, when for all $i{:}\,N;\ x{:}\,X_i$,

$$E_0(X;\phi)_i.x \ =_{H_i.X}\ E_1(X;\phi)_i.x\ . \tag{7.7}$$

Now, we prove that a final algebra with equations is obtained from a final algebra without equations by removing all elements that do not satisfy the equations and the elements that contain these elements.

Theorem 7.3 *If, for a functor $F{:}\,\mathcal{C} \to \mathcal{C}$ and law E, there exists a final F-coalgebra $(T;\delta)$, then the final $(F;E)$-coalgebra exists as well and is the greatest subalgebra of $(T;\delta)$ that satisfies law E, namely coalgebra $(T';\delta)$ where:*

$$T' \ :=\ \bigcup(X{:}\subseteq T\ |{:}\ \forall i{:}\,N;\ x{:}\in X_i\ ::\ \delta_i.x \in F_i.X \wedge (7.7)\ )$$

where the union and subset on tuples should be taken pointwise, and the functor is extended to subsets (par. 4.1.7).

Proof. We consider only the case $N = 1$, so we can forget about the subscripts i. $(T';\delta)$ is clearly an $(F;E)$-coalgebra. Let Ψ be another one; we must exhibit a unique homomorphism $\Psi \to (T';\delta)$ in $\mathbf{ALG}(F;E)$.
We have a unique F-homomorphism $f{:}\,\Psi \to (T;\delta)$, because $(T;\delta)$ is final. As E is a natural transformation and Ψ satisfies E, the range $f[\Psi]$ satisfies E too. So $f \in \Psi \to (T';\delta)$. As any other homomorphism to $(T';\delta)$ is a homomorphism to $(T;\delta)$ as well, it must equal f. ■

Actually, no really useful example of a final $(F;E)$-coalgebra is known to me.

7.5 Terminal interpretation of equations

In the tradition of "algebraic specification" [87], which we sketched in 4.7, one distinguishes between the initial and terminal interpretation of an algebraic specification. A specification consists of an algebra signature Σ together with a set of equations and sometimes inequations ($\neq$). Normally, only finitary signatures are allowed.

The *initial interpretation* of a specification is just the initial object in the category of all Σ-algebras that satisfy the equations. This corresponds to our notion of initial algebra with equations. Any inequations are superfluous: if they are not satisfied in this initial algebra, the specification is inconsistent.

The *terminal interpretation* of a specification is different, though. This is the final object in the category of all "term-generated" Σ-algebras that satisfy both the equations and inequations. An algebra Φ is *term-generated* iff the unique homomorphism from the initial Σ-algebra to Φ is surjective. Put otherwise, the terminal interpretation is the initial Σ-algebra modulo the greatest equivalence relation that satisfies the (in-)equations, when this exists. Thus, these terminal algebras are definitely not co-inductive types.

Presence of inequations is essential here; otherwise the terminal interpretation would trivially be the unit algebra. Normally, an algebraic specification includes some standard specification of one or more types whose elements are required to be distinct, like booleans, characters, or integers.

7.6 Conclusion

We described the dualization of inductive types, which are types with infinitely deep objects, for example indefinitely proceeding processes. The rigid form of recursive equation, needed to construct objects of these types, could be transformed into a more natural definition scheme.

The notion of algebra with equations can be dualized too, and is meaningful in the category of types, but this dual form of equation does not seem to be very useful.

Co-inductive types are not to be confused with the terminal interpretation of an algebraic specification.