Coalgebras as Types
Determined by their Elimination Rules

Anton Setzer
Swansea University (Wales, UK)

(Conference dedicated to Per Martin-Löf
on occasion of his retirement, May 5 - 8, 2009)

1. Inductive and coinductive types
2. Model for a type theory with coinductive types.
3. Meaning explanations – inductive and coinductive.

Prelim: Notation for Disjoint Union

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

  is the **disjoint union** of the elements nil' and $\mathrm{cons}'\ n\ x$ for $n : \mathbb{N}$ and $x : X$.

- So we have

$$\begin{array}{lcll} \mathrm{nil}' & : & \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, X) & , \\ \mathrm{cons}'\ n\ l & : & \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, X) \quad [n : \mathbb{N}, l : X] & . \end{array}$$

Notation for Disjoint Union

- And

$$\begin{array}{llll} t & := \mathrm{case}\ s\ \mathrm{of}\ \{(\mathrm{nil}') & \longrightarrow & case_{\mathrm{nil}} \\ & \qquad\qquad (\mathrm{cons}'\ n\ l) & \longrightarrow & case_{\mathrm{cons}}\ n\ l\} \end{array}$$

is the term s.t.

$$t = \begin{cases} case_{\mathrm{nil}} & \text{if } s = \mathrm{nil}' \quad , \\ case_{\mathrm{cons}}\ n\ l & \text{if } s = \mathrm{cons}'\ n\ l \quad . \end{cases}$$

1. Inductive and Coinductive Types

- Inductive types are determined by their **introduction rules**.
- Example List:

$$\begin{array}{lll} \mathrm{List} & : & \mathrm{Set} \\ \mathrm{nil} & : & \mathrm{List} \\ \mathrm{cons} & : & (n : \mathbb{N}, l : \mathrm{List}) \longrightarrow \mathrm{List} \end{array}$$

- Elimination rules express List is the least set introduced by those introduction rules:

Elimination Rules for List

- Assume

$$\begin{array}{lcl} A & : & \mathrm{List} \longrightarrow \mathrm{Set} \\ step_{\mathrm{nil}} & : & A\ \mathrm{nil}\ \ , \\ step_{\mathrm{cons}} & : & (n : \mathbb{N}, l : \mathrm{List}, ih : A\ l) \longrightarrow A\ (\mathrm{cons}\ n\ l) \end{array}$$

  Then we have

$$f := \mathrm{elim}\ A\ step_{\mathrm{nil}}\ step_{\mathrm{cons}} : (l : \mathrm{List}) \longrightarrow A\ l$$

$$\begin{array}{lcl} f\ \mathrm{nil} & = & step_{\mathrm{nil}} \\ f\ (\mathrm{cons}\ n\ l) & = & step_{\mathrm{cons}}\ n\ l\ (f\ l) \end{array}$$

List as an Initial Algebra

- The above rules correspond to List being the **initial algebra** for the functor

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

- That List is an algebra for F means that we have a function

$$\mathrm{intro} : (\mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, \mathrm{List})) \to \mathrm{List}$$

- From this we obtain the **introduction rules** for List:

$$\begin{array}{lcllll} \mathrm{nil} & := & \mathrm{intro}\ \mathrm{nil}' & : & \mathrm{List} & , \\ \mathrm{cons}\ n\ l & := & \mathrm{intro}\ (\mathrm{cons}'\ n\ l) & : & \mathrm{List} & . \end{array}$$

- So **formation/introduction rules** express List is an **algebra for F**.

List as an Initial Algebra

- That List is an **initial algebra** means that if f is as below there exists a **unique** $g := \mathrm{elim}\ f$ s.t. the following commutes:

$$
\begin{array}{ccc}
\mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, \mathrm{List}) & \xrightarrow{\mathrm{intro}} & \mathrm{List} \\
\Big\downarrow {\scriptstyle \mathrm{nil}' + \mathrm{cons}'(\mathrm{id}_{\mathbb{N}}, g)} & & \Big\downarrow {\scriptstyle g} \\
\mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, X) & \xrightarrow{f} & X
\end{array}
$$

- Let

$$
\begin{array}{lcl}
step_{\mathrm{nil}} & = & f\ \mathrm{nil}' \\
step_{\mathrm{cons}}\ n\ l & = & f\ (\mathrm{cons}'\ n\ l)
\end{array}
$$

List as an Initial Algebra

$$\begin{array}{ccc} \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, \mathrm{List}) & \xrightarrow{\mathrm{intro}} & \mathrm{List} \\ \downarrow{\scriptstyle \mathrm{nil}' + \mathrm{cons}'(\mathrm{id}_{\mathbb{N}}, g)} & & \downarrow{\scriptstyle g} \\ \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, X) & \xrightarrow{[step_{\mathrm{nil}}, step_{\mathrm{cons}}]} & X \end{array}$$

$$\begin{array}{lcl} g\ \mathrm{nil} & = & step_{\mathrm{nil}} \\ g\ (\mathrm{cons}\ n\ l) & = & step_{\mathrm{cons}}\ n\ (g\ l) \end{array}$$

- We obtain **iteration** and can derive the principle of **dependent higher type prim. recursion** from **uniqueness** of iteration.

Colist

- Let coList be the **weakly final coalgebra** for F.
- coList is a **coalgebra** means that we have

$$\_.\mathrm{unfold} : \mathrm{coList} \to (\mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, \mathrm{coList}))$$

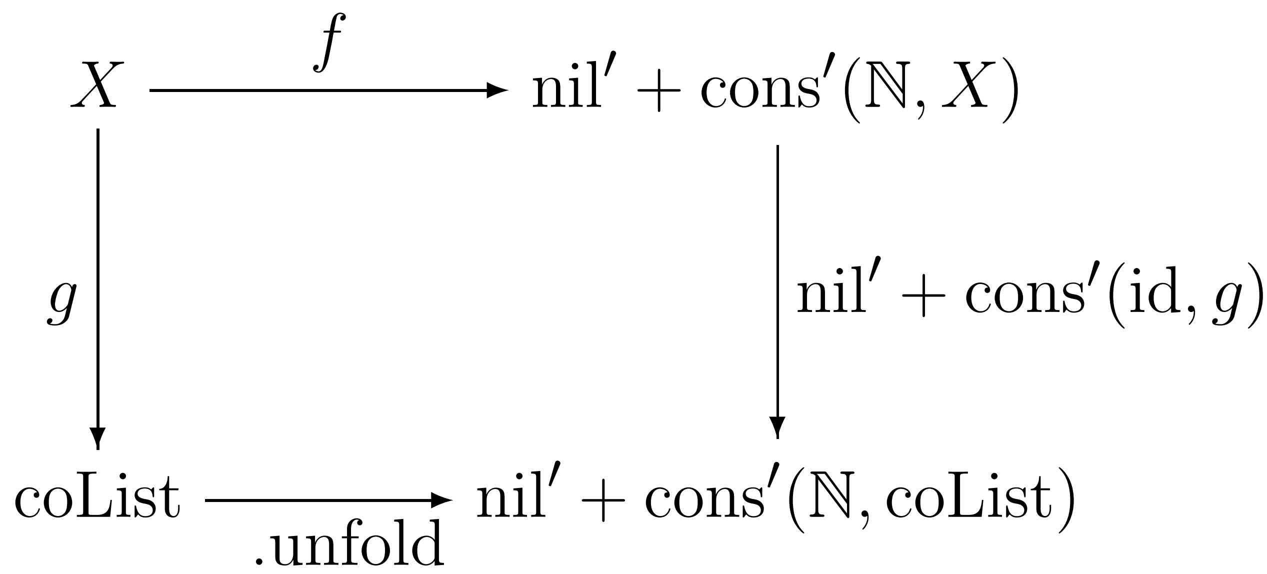
(used as postfix operation).

- So, if $l : \mathrm{coList}$ then

$$\begin{array}{lcl} l.\mathrm{unfold} & = & \mathrm{nil}' \qquad\qquad \text{or} \\ l.\mathrm{unfold} & = & \mathrm{cons}'\ n\ l \end{array}$$

Colist as Weakly Final Coalgebra

- Assume f as below. Then there exists g (called $\mathrm{intro}\ X\ f$) s.t.



- So

$$
\begin{array}{l}
(g\ x).\mathrm{unfold} = \\
\qquad \mathrm{case}\ (f\ x)\ \mathrm{of}\ \{(\mathrm{nil}') \quad \longrightarrow \quad \mathrm{nil}' \\
\qquad \qquad \qquad \qquad (\mathrm{cons}'\ n\ y) \quad \longrightarrow \quad \mathrm{cons}'\ n\ (g\ y)
\end{array}
$$

Guarded Recursion

$$
\begin{array}{l}
(g\ x).\mathrm{unfold} = \\
\qquad \mathrm{case}\ (f\ x)\ \mathrm{of}\ \{(\mathrm{nil}') \qquad\qquad \longrightarrow\ \mathrm{nil}' \\
\qquad\qquad\qquad\qquad\qquad (\mathrm{cons}'\ n\ y) \longrightarrow\ \mathrm{cons}'\ n\ (g\ y)\}
\end{array}
$$

- f contains the information needed to define a simple but generic case of **guarded recursion**:
  We can define $f : X \to \mathrm{coList}$ s.t.

$$
\begin{array}{lcll}
(f\ x).\mathrm{unfold} & = & \mathrm{nil}' & \text{or} \\
(f\ x).\mathrm{unfold} & = & \mathrm{cons}'\ n\ (f\ y) & \text{for some } n, y
\end{array}
$$

- More general cases of guarded recursion can be derived assuming a final coalgebra.

Example

- Define

$$\mathrm{inc} : \mathbb{N} \longrightarrow \mathrm{coList}$$
$$(\mathrm{inc}\ n).\mathrm{unfold} = \mathrm{cons}'\ n\ (\mathrm{inc}\ (n+1))$$

- Roughly speaking, $\mathrm{inc}\ n$ is

$$\mathrm{cons}'\ n\ (\mathrm{cons}'\ (n+1)\ (\mathrm{cons}'\ (n+2)\cdots))$$

But the unfolding is controlled by unfold, which avoids non-normalisation.

Other examples of Coalgebras

- **Interactive programs** as coalgebras:
- Let

$$\begin{array}{l}\text{coalg IO} : \text{Set where} \\ \quad \begin{array}{lcl} \_.\text{command} & : & \text{IO} \longrightarrow (\text{read} + \text{write}(\text{String})) \\ \_.\text{next} & : & (p : \text{IO}) \longrightarrow \text{R}\ (\text{command}\ p) \end{array}\end{array}$$

  where

$$\begin{array}{lcl} \text{R read} & = & \text{String} \rightarrow \text{IO} \\ \text{R}\ (\text{write}\ s) & = & \text{IO} \end{array}$$

- $\_.\text{unfold}$, $\_.\text{command}$, $\_.\text{next}$ above are **destructors**, dual of a **constructor**.

Example Program

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathrm{mutual}} \\
\quad \mathrm{echo}_0 & : & \mathrm{IO} \\
\quad \mathrm{echo}_0.\mathrm{command} & = & \mathrm{read} \\
\quad \mathrm{echo}_0.\mathrm{next}\ s & = & \mathrm{echo}_1\ s \\
 & & \\
\quad \mathrm{echo}_1 & : & \mathrm{String} \longrightarrow \mathrm{IO} \\
\quad (\mathrm{echo}_1\ s).\mathrm{command} & = & \mathrm{write}\ s \\
\quad (\mathrm{echo}_1\ s).\mathrm{next} & = & \mathrm{echo}_0
\end{array}
$$

Failure of Logic in Computer Scienc

- Mark Priestley in a talk in Swansea:
  - **Failure of logic to contribute to computer science.**
    - Substantial contribution of logic to development of Algol.
    - Since emergence of object-oriented programming **lead of development taken by practical computing**.
  - Possible explanation (A. S.):
    - Programs in computer science switched from **batch programs** to **interactive programs**.
    - Interactive programs correspond to **coinductive** rather than **inductive definitions**.
    - Coinductive definitions underdeveloped in logic.

Other examples of Coalgebras

- The set of real numbers in $[-1, 1]$ having a **binary expansion**:

$$r = 0.d_0 d_1 d_2 \cdots$$

with $d_i \in \{-1, 0, 1\}$
is given by

$$\begin{array}{lll}
\multicolumn{3}{l}{\mathrm{coalg}\ \mathcal{R} : \mathbb{R} \to \mathrm{Set}\ \mathrm{where}} \\
\quad _.\mathrm{p} & : & \{r : \mathbb{R}\} \to (q : \mathcal{R}\ r) \to r \in [-1, 1] \\
\quad _.\mathrm{digit} & : & \{r : \mathbb{R}\} \to (q : \mathcal{R}\ r) \to \{-1, 0, 1\} \\
\quad _.\mathrm{tail} & : & \{r : \mathbb{R}\} \to (q : \mathcal{R}\ r) \to \mathcal{R}\ (2\ \cdot\ r - q.\mathrm{digit})
\end{array}$$

($\{r : \mathbb{R}\}$ is a hidden argument.)

Binary expansion of $\frac{1}{3}$

- E.g.

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\mathrm{mutual}} \\
\quad q_0 & : & \mathcal{R}\ \frac{1}{3} \\
\quad q_0.\mathrm{p} & = & \cdots : \frac{1}{3} \in [-1, 1] \\
\quad q_0.\mathrm{digit} & = & 0 \\
\quad q_0.\mathrm{tail} & = & q_1 : \mathcal{R}\ (2 \cdot \frac{1}{3} - 0) \\
 & & \\
\quad q_1 & : & \mathcal{R}\ \frac{2}{3} \\
\quad q_1.\mathrm{p} & = & \cdots : \frac{2}{3} \in [-1, 1] \\
\quad q_1.\mathrm{digit} & = & 1 \\
\quad q_1.\mathrm{tail} & = & q_0 : \mathcal{R}\ (2 \cdot \frac{2}{3} - 1)
\end{array}
$$

Informal Treatment of the above

- The set of real numbers with binary expansion is **coinductively** defined as follows:
  - If $r \in [-1, 1]$, and there exists a $\mathrm{digit} \in \{-1, 0, 1\}$ s.t. $2 \cdot r - \mathrm{digit}$ has a binary expansion, then r has a binary expansion.
- We prove that $\frac{1}{3}$ and $\frac{2}{3}$ have binary expansion simultaneously:
  - Both are in $[-1, 1]$.
  - For $r := \frac{1}{3}$ we have with $\mathrm{digit}_0 := 0$ that $2 \cdot r - \mathrm{digit}_0$ has a binary expansion by **co-IH**.
  - For $r := \frac{2}{3}$ we have with $\mathrm{digit}_1 := 1$ that $2 \cdot r - \mathrm{digit}_1$ has a binary expansion by **co-IH**.

(Bi)simulation

- Let an A-labelled transition system $\mathrm{T}_1, \longrightarrow$ be given by $\mathrm{T}_1 : \mathrm{Set}$ and a relation $\mathrm{R} \subseteq \mathrm{T}_1 \times A \times \mathrm{T}_1$ written as $t \stackrel{a}{\longrightarrow} t'$ for $(t, a, t') \in \mathrm{R}$.
- Let T_1, T_2 be A-labelled transition system.
- Simulation between T_1 and T_2 is the largest relation $\leq \subseteq \mathrm{T}_1 \times \mathrm{T}_2$ s.t.

$$\forall t_1 \in \mathrm{T}_1.\forall t_2 \in \mathrm{T}_2.\forall a \in A.\forall t_1' \in \mathrm{T}_1.t_1 \leq t_2 \rightarrow t_1 \stackrel{a}{\longrightarrow} t_1' \rightarrow \exists t_2 \in \mathrm{T}_2.t_2 \stackrel{a}{\longrightarrow} t_2' \wedge t_1' \leq t_2'$$

(Bi)simulation

- Defined as a coalgebra as

$$\begin{aligned}
&\text{coalg} \leq : \mathrm{T}_1 \to \mathrm{T}_2 \to \mathrm{Set} \text{ where}\\
&\quad _.\text{unfold} : \{t_1 : \mathrm{T}_1, t_2 : \mathrm{T}_2\} \to (t_1 \leq t_2, t'_1 : \mathrm{T}_1, a : A, t_1 \xrightarrow{a} t'_1\\
&\qquad\qquad \to (t'_2 : \mathrm{T}_2) \times (t_2 \xrightarrow{a} t'_2) \times (t'_1 \leq t'_2)
\end{aligned}$$

Example

- Let T_1 be given as

$$a_1 \xrightarrow{\text{tick}} a_2 \xrightarrow{\text{tick}} a_3 \xrightarrow{\text{tick}} \cdots$$

- Let $\mathrm{T}_1 = \{a\}$ with $a \xrightarrow{tick} a$.
- Traditional proof of $\forall n : \mathbb{N}.a_n \leq a$:
  - Let $R := \{\langle a_n, a\rangle \mid n \in \mathbb{N}\}$.
  - Show R is a simulation relation.
  - Need to show that if $\langle a_n, a\rangle \in R$, $a_n \xrightarrow{x} a'$, then there exists a'' s.t. $\langle a', a''\rangle \in R$ and $a \xrightarrow{x} a''$.
  - Now in the above situation we have $x = \text{tick}$, $a' = a_{n+1}$.
  - Let $a'' := a$, then the conditions are fulfilled.
  - So R is a simulation relation, $R \subseteq \leq$, so $a_n \leq a$.

Example

- Proof of $\forall n \in \mathbb{N}.a_n \leq a$ using guarded recursion:

$$\begin{aligned}
&\mathrm{lem} : (n : \mathbb{N}) \to a_n \leq a \\
&(\mathrm{lem}\ n).\mathrm{unfold}\ a_{n+1}\ \mathrm{tick}\ \underbrace{\mathrm{triv}}_{:a_n \xrightarrow{\mathrm{tick}} a_{n+1}} = \langle a, \mathrm{triv}, \mathrm{lem}\ (n+1) \rangle
\end{aligned}$$

Informal Reading of the Proof

- We show $\forall n \in \mathbb{N}.a_n \leq a$ by coinduction on $a_n \leq a$:
  - Assume $a_n \xrightarrow{x} a'$.
  - Then $x = \text{tick}$, $a' = a_{n+1}$.
  - Then we have $a \xrightarrow{x} a$ and by coIH $a_{n+1} \leq a$.
  - So $a_n \leq a$.

2. Model

- For simplicity we ignore here equalities.
- Sets modelled as sets of terms.
- **Model of List**:
  - $[\![\mathrm{List}]\!]$ is defined inductively by:
    - If $t \longrightarrow \mathrm{nil}$, then $t \in [\![\mathrm{List}]\!]$.
    - If $n \in [\![\mathbb{N}]\!]$, $l \in [\![\mathrm{List}]\!]$, $t \longrightarrow \mathrm{cons}\ n\ l$, then $t \in [\![\mathrm{List}]\!]$.

Model of $A \to B$

- $[\![A \to B]\!] := \{f \mid \forall a \in [\![A]\!].f\ a \in [\![B]\!]\}$.

Model of coList

- Define for (Meta-)$n \in \mathbb{N}$

$$
\begin{array}{lcl}
\mathrm{red}_n & : & \mathrm{Term} \rightarrow_{\mathrm{partial}} \mathrm{Term} \\
\mathrm{red}_0\ a & := & a.\mathrm{unfold} \\
\mathrm{red}_{n+1}\ a & := & \begin{cases} \mathrm{nil}' & \text{if } \mathrm{red}_n\ a \longrightarrow \mathrm{nil}' \\ a'.\mathrm{unfold} & \text{if } \mathrm{red}_n\ a \longrightarrow \mathrm{cons}'\ n\ a' \\ \text{undefined} & \text{otherwise.} \end{cases}
\end{array}
$$

- Now

$$
\begin{array}{l}
[\![\,\mathrm{coList}\,]\!] := \{t \mid \forall n \in \mathbb{N}.\mathrm{red}_n\ t \downarrow \wedge \\
\qquad (\mathrm{red}_n\ t \longrightarrow \mathrm{nil}' \vee \\
\qquad \exists m \in [\![\,\mathbb{N}\,]\!].\exists t.\mathrm{red}_n\ t \longrightarrow \mathrm{cons}'\ m\ t)\}
\end{array}
$$

Definition using largest Fixed points

- The above definition avoids the use of largest fixed points.
- Using largest fixed points we can define

$$\begin{aligned}[\![\,\mathrm{coList}\,]\!] = \text{largest set } X \text{ s.t.}\\ &\forall t \in X.\ \ t.\mathrm{unfold} \longrightarrow \mathrm{nil}' \vee \\ &\quad \exists n \in [\![\,\mathbb{N}\,]\!].t' \in X.\ \ t.\mathrm{unfold} \longrightarrow \mathrm{cons}'\ n\ t'\end{aligned}$$

- or: **Coinductive definition of $[\![\,\mathrm{coList}\,]\!]$**:
  If
  - $t.\mathrm{unfold} \longrightarrow \mathrm{nil}'$ or
  - $t.\mathrm{unfold} \longrightarrow \mathrm{cons}'\ n\ t'$ some $n \in [\![\,\mathbb{N}\,]\!]$, $t' \in [\![\,\mathrm{coList}\,]\!]$

  then $t \in [\![\,\mathrm{coList}\,]\!]$.

3. Meaning Explanations

- Meaning explanations of inductive data types correspond to the **introduction rules**.
- E.g. Meaning of List:
  - nil is a **canonical element** of List.
  - If n is a natural number, l is a (not necessarily canonical) element of List, then $\mathrm{cons}\ n\ l$ is a **canonical element** of List.
  - An arbitrary element of List is a program which **evaluates** to a canonical element of List.
- Meaning of nil and cons is trivial (they compute canonical elements of List).
- Meaning of the terms introduced by the elimination rules refers to the meaning of elements of List.

Example: Explanation of append

- Let

$$\begin{array}{lcl}\mathrm{append} & : & \mathrm{List} \to \mathrm{List} \to \mathrm{List} \\ \mathrm{append}\ \mathrm{nil}\ l' & = & l' \\ \mathrm{append}\ (\mathrm{cons}\ n\ l)\ l' & = & \mathrm{cons}\ n\ (\mathrm{append}\ l\ l')\end{array}$$

- We show how to compute for l, l' elements of List $\mathrm{append}\ l\ l'$, which is an element of List:
  $\mathrm{append}\ l\ l'$ is computed as follows:
  - Compute l. We obtain either nil or and $\mathrm{cons}\ n\ \ l''$ for an element n of $\mathbb{N}$ and an element l'' introduced before l.

Explanation of append

- If we obtain nil, the result of the computation is l' which is an element of List.
- If we obtain $\mathrm{cons}\ n\ l''$ we know how to compute $\mathrm{append}\ l''\ l'$ which is an element of List.
  The program evaluates to $\mathrm{cons}\ n\ (\mathrm{append}\ l''\ l)$ which is a (canonical) element of List.

Meaning of $A \to B$

- We give the meaning of the logical framework $A \to B$ (not of $\Pi x : A.B$).
- An element f of $A \to B$ is a program, which, taken as input an element of A, computes an element of B.
- Meaning of the result of the elimination rule for $A \to B$:

$$\frac{f : A \to B \qquad a : A}{f\ a : B}$$

Assume $f : A \to B$. Then f is a program which computes from any element a of A an element of B. $f\ a$ evaluates to the element computed by f from input a.

Meaning of $A \to B$

- We give an explanation of the term introduced by the introduction rule for $A \to B$:

$$\frac{x : A \Rightarrow b : B}{(x)b : A \to B}$$

  - Assume we have depending on a hypothetical element x of A that b is an element of B.
  - Then $(x)b$ is the element $A \to B$, which if evaluated with input a which is an element of A evaluates to $b[x := a]$, which is an element of B.

Meaning of $A \to B$

- So the meaning of $A \to B$ is given by its **elimination rule**.
- The meaning of the terms introduced by the introduction rule is given by referring to this definition.

Meaning of coList

- Meaning of coList:
  - An element of coList is a program which can **take as input an unfold operation**. If it receives this operation, it computes to either nil' or $\mathrm{cons}'\ n\ l$ for an element n of $\mathbb{N}$ and an element l of coList.
- $l.\mathrm{unfold}$ for l an element of coList is the program computed by l if receiving an unfold.
- As an example of the meaning explanations for the introduction rules, we give the meaning of $\mathrm{inc}\ n$ for n an element of $\mathbb{N}$, where inc is defined as follows:

$$\mathrm{inc} : \mathbb{N} \longrightarrow \mathrm{coList}$$

$$(\mathrm{inc}\ n).\mathrm{unfold} = \mathrm{cons}'\ n\ (\mathrm{inc}\ (n+1))$$

Meaning of coList

- We need to show that $\mathrm{inc}\ n$ is an element of coList for any n of $\mathbb{N}$.
  - $\mathrm{inc}\ n$ is the program, which, if it receives an unfold call, evaluates to $\mathrm{cons}'\ n\ (\mathrm{inc}\ (n+1))$.
  - n is an element of $\mathbb{N}$.
  - $\mathrm{inc}\ (n+1)$ is an element of coList.

Conclusion

- Coalgebras = **dual of algebras**.
- In mathematics long tradition of **inductive definitions**, and proofs by induction.
- **Coinductive definitions** and proofs by coinduction (= guarded recursion) **not much used**.
- In computer science coinductive definitions are **as important as inductive definitions**.

Conclusion

- Meaning for inductive types are given by their **introduction rules**.
  - Explanation of terms given by **elimination rules** is introduced in a **second step**.
- Meaning for the function type and for coinductive types (coalgebras) are given by their **elimination rules**.
  - Meaning of terms given by **introduction rules**n is defined in a **second step**.
- So some types are **determined by how we introduce them**, and some are **determined** by **what we can do with its elements**.

Conclusion

- We loose the notion of a **canonical element** of a set.
- There are introduction rules for elements of a weakly final coalgebra.
- An element of a weakly final coalgebra C for functor F reduces to $\mathrm{intro}\ X\ f$ where $f : X \to F(X)$.
  - That's similar to an element of $A \to B$ reducing to something of the form $(x)b$ for $x : A \Rightarrow b : B$.