Coalgebras and Codata in Agda

Anton Setzer
Swansea University (Wales, UK)

(Wessex Seminar, Bath, 3 March 2009)

1. The concept of codata.
2. Codata in Agda.
3. Weakly Final Coalgebras in Dependent Type Theory.
4. Proofs by Corecursion.

1. The Concept of “Codata”

- “codata” introduced in functional languages as a data type of infinite objects.
- “data” corresponds to well-founded objects e.g.

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

- Elements of List are finite lists e.g.

$$\mathrm{cons}\ 3\ (\mathrm{cons}\ 4\ \mathrm{nil})$$

Induction over data

- Therefore we can define operations by recursion on lists e.g.

$$
\begin{array}{lcl}
\mathrm{length} : \mathrm{List} \to \mathbb{N} & & \\
\mathrm{length}\ \mathrm{nil} & = & 0 \\
\mathrm{length}\ (\mathrm{cons}\ n\ l) & = & \mathrm{length}\ l + 1
\end{array}
$$

CoList

- If we use codata we have

$$\begin{array}{l}\mathrm{codata}\ \mathrm{coList} : \mathrm{Set}\ \mathrm{where} \\ \qquad \begin{array}{lll} \mathrm{nil} & : & \mathrm{coList} \\ \mathrm{cons} & : & \mathbb{N} \to \mathrm{coList} \to \mathrm{coList} \end{array}\end{array}$$

- coList contains infinite objects, e.g.

$$\begin{array}{l}\omega : \mathrm{coList} \\ \omega = \mathrm{cons}\ 0\ (\mathrm{cons}\ 0\ (\mathrm{cons}\ 0\ \cdots))\end{array}$$

- We can define ω by coiteration or guarded recursion:

$$\begin{array}{l}\omega : \mathrm{coList} \\ \omega = \mathrm{cons}\ 0\ \omega\end{array}$$

Guarded Recursion

- We can not define length anymore but $\mathrm{colength}$ by coiteration **into** $\mathrm{co}\mathbb{N}$

$$\begin{array}{l} \mathrm{codata}\ \mathrm{co}\mathbb{N} : \mathrm{Set}\ \mathrm{where} \\ \quad \begin{array}{lcl} 0 & : & \mathrm{co}\mathbb{N} \\ \mathrm{S} & : & \mathrm{co}\mathbb{N} \longrightarrow \mathrm{co}\mathbb{N} \end{array} \end{array}$$

$$\begin{array}{lcl} \multicolumn{3}{l}{\mathrm{colength} : \mathrm{coList} \longrightarrow \mathrm{co}\mathbb{N}} \\ \mathrm{colength}\ \mathrm{nil} & = & 0 \\ \mathrm{colength}\ (\mathrm{cons}\ n\ l) & = & \mathrm{S}\ (\mathrm{colength}\ l) \end{array}$$

- So

$$\mathrm{colength}\ \omega = \mathrm{S}\ (\mathrm{S}\ (\mathrm{S}\ \cdots)))$$

Problem

- Problem of this approach: undecidability of equality.
- For
$$f, g : \mathbb{N} \to \mathbb{N}$$
we can define by coiteration $l_f, l_g : \mathrm{coList}$ s.t.
$$\begin{array}{lcl} l_f & = & \mathrm{cons}(f\ 0)\ (\mathrm{cons}\ (f\ 1)\ (\mathrm{cons}\ (f\ 2) \cdots)) \\ l_g & = & \mathrm{cons}(g\ 0)\ (\mathrm{cons}\ (g\ 1)\ (\mathrm{cons}\ (g\ 2) \cdots)) \end{array}$$
- l_f and l_g are equal if $f = g$.
- Equality on $\mathbb{N} \to \mathbb{N}$ is undecidable, therefore on coList as well.
- Type checking for dependently typed language requires equality checking, therefore type checking becomes undecidable.

Intensional Equality

- Two functions are equal if their programs have the same normal form:
  - $\lambda\, x.s = \lambda\, x.t$ if s and t have the same normal form.
- In the same way we can only achieve that two elements of coList are equal, if the programs for forming them are equal.

Example IO

- Assume
  - $\mathrm{C} : \mathrm{Set}$ set of commands
  - $\mathrm{R} : \mathrm{C} \to \mathrm{Set}$ set of responses to a command.
- Example
  - $\mathrm{C} = \mathrm{read} + \mathrm{write}(s : \mathrm{String}) + \mathrm{terminate}$.
  - $\mathrm{R}\ \mathrm{read} = \mathrm{String}$,
  - $\mathrm{R}\ (\mathrm{write}\ s) = \{*\}$,
  - $\mathrm{R}\ \mathrm{terminate} = \emptyset$.

Example IO

$$\text{codata IO } (\mathrm{C} : \mathrm{Set})(\mathrm{R} : \mathrm{C} \rightarrow \mathrm{Set}) : \mathrm{Set} \text{ where}$$
$$\quad \text{prog} : (c : \mathrm{C}) \rightarrow (n : \mathrm{R}\ c \rightarrow \mathrm{IO\ C\ R}) \rightarrow \mathrm{IO\ C\ R}$$

$$p : \mathrm{IO\ C\ R}$$
$$p = \text{prog (write "Password: ")}$$
$$(\lambda\ \_.(\text{prog read}$$
$$(\lambda\ passwd.\text{if } passwd = \text{"1234"}$$
$$\text{then (prog terminate efq)}$$
$$\text{else (prog (write "Wrong Password!")}$$
$$(\lambda\ \_.p)))))$$

Objects

$$
\begin{array}{l}
\text{class Cell}\{ \\
\qquad n : \mathbb{N}; \\
\qquad \text{set } (m : \mathbb{N}) : \text{void } \{ \\
\qquad\qquad n = m; \}; \\
\qquad \text{get } () : \mathbb{N} \ \{ \\
\qquad\qquad \text{return } n; \}; \}
\end{array}
$$

Modelling Cell using Codata

$$
\begin{aligned}
&\text{codata Cell} : \text{Set where} \\
&\qquad \text{createCell} : (\mathit{set} : \mathbb{N} \longrightarrow \text{Cell}) \\
&\qquad\qquad\qquad\qquad \longrightarrow (\mathit{get} : \mathbb{N} \times \text{Cell}) \\
&\qquad\qquad\qquad\qquad \longrightarrow \text{Cell}
\end{aligned}
$$

$$
\begin{aligned}
&\text{cell} : \mathbb{N} \longrightarrow \text{Cell} \\
&\text{cell } n = \text{createCell } (\lambda\, m.\text{cell } m) \\
&\qquad\qquad\qquad\qquad\quad \langle n, \text{cell } n \rangle
\end{aligned}
$$

2. Codata in Agda

- Use of codata type.
- However, we do not have extensional equality for codata types.
- Guarded recursion defined using “$\sim$”:
- Example code:

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

$$\begin{array}{l} \omega : \text{coList} \\ \omega \;\sim\; \text{cons}\ 0\ \omega \end{array}$$

After one unfolding ω and $\text{cons}\ 0\ \omega$ are the same.

Pattern Matching

- Case distinction on codata types defined using pattern matching:

$$\begin{array}{lcl} f : \mathrm{coList} \to \mathbb{N} & & \\ f\ \mathrm{nil} & = & 0 \\ f\ (\mathrm{cons}\ n\ l) & = & n \end{array}$$

- Example:

$$\begin{array}{l} \mathrm{codata}\ \mathrm{co}\mathbb{N} : \mathrm{Set}\ \mathrm{where} \\ \quad \begin{array}{lcl} 0 & : & \mathrm{co}\mathbb{N} \\ \mathrm{S} & : & \mathrm{co}\mathbb{N} \to \mathrm{co}\mathbb{N} \end{array} \\ f : \mathrm{coList} \to \mathrm{co}\mathbb{N} \\ \begin{array}{lcl} f\ \mathrm{nil} & \sim & 0 \\ f\ (\mathrm{cons}\ 0\ l) & \sim & \mathrm{S}\ (f\ l) \\ f\ (\mathrm{cons}\ (\mathrm{S}\ n)\ l) & \sim & \mathrm{S}\ (f\ n\ l) \end{array} \end{array}$$

Problem of Subject Reduction

- Consider the following code (by Nicolas Ory; same problem occurs in Coq):

$$\text{data } \_ == \_ \; (x : \text{coList}) : \text{coList} \to \text{Set where}$$
$$\;\text{refl} : a == a$$

$$\text{out} : \text{coList} \to \text{coList}$$
$$\begin{array}{lcl} \text{out nil} & = & \text{nil} \\ \text{out } (\text{cons } n \; l) & = & \text{cons } n \; l \end{array}$$

$$\text{lemma} : (l : \text{coList}) \to l == \text{out } l$$
$$\begin{array}{lcl} \text{lemma nil} & = & \text{refl} \\ \text{lemma } (\text{cons } n \; l) & = & \text{refl} \end{array}$$

$$\text{p} : \omega == \text{cons } 0 \; \omega$$
$$\text{p} = \text{lemma } \omega$$

Problem of Subject Reduction

$$\begin{aligned} \mathrm{p} &: \omega == \mathrm{cons}\ 0\ \omega \\ \mathrm{p} &= \mathrm{lemma}\ \omega \end{aligned}$$

- $\mathrm{p} \longrightarrow \mathrm{refl}$ but we don't have $\mathrm{refl} : \omega == \mathrm{cons}\ 0\ \omega$.
- Quick fix in Agda:
  Dependent pattern matching on codata is not allowed.
  Therefore the code

$$\begin{array}{lcl} \multicolumn{3}{l}{\mathrm{out} : \mathrm{coList} \to \mathrm{coList}} \\ \mathrm{out}\ \mathrm{nil} & = & \mathrm{nil} \\ \mathrm{out}\ (\mathrm{cons}\ n\ l) & = & \mathrm{cons}\ n\ l \end{array}$$

causes an error.

Underlying Problem

- Unclear what “$\sim$” means.
- Unclear what pattern matching on coalgebras means

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\mathrm{out} : \mathrm{coList} \to \mathrm{coList}} \\
\mathrm{out}\ \mathrm{nil} & = & \mathrm{nil} \\
\mathrm{out}\ (\mathrm{cons}\ n\ l) & = & \mathrm{cons}\ n\ l
\end{array}
$$

For which l does one of the above patterns trigger?

3. Weakly Final Coalgebras in Dept.

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

    stands for

    $$\{*\} + \mathbb{N} \times X$$

    with the following definitions:

    $$\begin{array}{lcl} \mathrm{nil}' & = & \mathrm{inl}\ * \\ \mathrm{cons}'\ n\ l & = & \mathrm{inr}\ \langle n, l \rangle \end{array}$$

Coalgebras in Category Theory

- A coalgebra for the functor $\mathrm{F} : \mathrm{Set} \longrightarrow \mathrm{Set}$

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

is an arrow

$$\mathrm{coList} \xrightarrow{\mathrm{case}} \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, \mathrm{coList})$$

- We do no longer have for $l : \mathrm{coList}\ l = \mathrm{nil}'$ or $l = \mathrm{cons}'\ n\ l$.
- Instead we have that for $l : \mathrm{coList}$

$$\mathrm{case}\ l = \mathrm{nil}' \text{ or } \mathrm{case}\ l = \mathrm{cons}'\ n\ l'$$

- So elements of coList are not infinite but might be unfolded infinitely often using case.

Coalgebras in Category Theory

$$\begin{tikzcd}
\mathrm{coList} \arrow[r, "\mathrm{case}"] & \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, \mathrm{coList}) \\
A \arrow[u, "\exists g"] \arrow[r, "f"] & \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, A) \arrow[u, "\mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, g)"']
\end{tikzcd}$$

- If $f(a) = \mathrm{nil}'$, then $\mathrm{case}\ (g\ a) = \mathrm{nil}'$.
- If $f(a) = \mathrm{cons}'\ n\ a'$, then $\mathrm{case}\ (g\ a) = \mathrm{cons}'\ n\ (g\ a)$.

Coalgebras in Category Theory

- If $f(a) = \mathrm{nil}'$, then $\mathrm{case}\ (g\ a) = \mathrm{nil}'$.
- If $f(a) = \mathrm{cons}'\ n\ a'$, then $\mathrm{case}\ (g\ a) = \mathrm{cons}'\ n\ (g\ a)$.

- The above allows to define

$$g : A \longrightarrow \mathrm{coList}$$

$$\mathrm{case}\ (g\ a) = \mathrm{nil}'$$

or

$\mathrm{cons}'\ n\ (g\ a')$ for some n, a'

Dual of the Constructors

- Using bisimulation equality we can derive

$$\begin{aligned}&\mathrm{nil} : \mathrm{coList}\\&\mathrm{case}\ \mathrm{nil} = \mathrm{nil}'\end{aligned}$$

$$\begin{aligned}&\mathrm{cons} : \mathbb{N} \to \mathrm{coList} \to \mathrm{coList}\\&\mathrm{case}\ (\mathrm{cons}\ n\ l) = \mathrm{cons}'\ n\ l\end{aligned}$$

- Note that

$$\mathrm{cons}'\ n\ l : \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, \mathrm{coList})$$

whereas

$$\mathrm{cons}\ n\ l : \mathrm{coList}$$

Corecursion

- We can extend the principle of guarded recursion to allow such definitions:

$$\begin{tikzcd}
\mathrm{coList} \arrow[r, "\mathrm{case}"] & \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, \mathrm{coList}) \\
A \arrow[u, "\exists g"] \arrow[r, "f"] & \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, A) + \mathrm{cons}'(\mathbb{N}, \mathrm{coList}) \arrow[u, "\mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, g) + \mathrm{cons}'(\mathbb{N}, \mathrm{id})"']
\end{tikzcd}$$

- The dual of the step from iteration to recursion, which allows to define by recursion

$$\begin{array}{lcl}
\multicolumn{3}{l}{\mathrm{pred} : \mathbb{N} \longrightarrow \mathbb{N}} \\
\mathrm{pred}\ 0 & = & 0 \\
\mathrm{pred}\ (\mathrm{S}\ n) & = & n
\end{array}$$

Deep Corecursion

- We can extend corecursion further in order to allow

$$g : A \longrightarrow \mathrm{coList}$$

$$\mathrm{case}\ (g\ a) = \mathrm{nil}'$$

or

$$\mathrm{cons}'\ n_1\ (\mathrm{cons}\ n_2\ (\cdots (\mathrm{cons}\ n_k\ (g\ a)) \cdots))$$

for some $k \geq 1, n_i, a$

or

$$\mathrm{cons}'\ n_1\ (\mathrm{cons}\ n_2\ (\cdots (\mathrm{cons}\ n_k\ l) \cdots))$$

for some $k \geq 1, n_i, l$

- Dual of course of value induction.

Formal Diagram

- Let F^{∞} the weakly final coalgebra for F.
- Let $\mathrm{build}_n : \mathrm{F}^n(\mathrm{F}^{\infty}) \to \mathrm{F}^{\infty}$ the n-times build.

$$
\begin{tikzcd}
\mathrm{F}^{\infty} \arrow[r, "\mathrm{case}"] & \mathrm{F}(\mathrm{F}^{\infty}) \\
 & \Sigma_{n:\mathbb{N}}\mathrm{F}(\mathrm{F}^n(\mathrm{F}^{\infty})) \arrow[u, "\Sigma_{n:\mathbb{N}}\mathrm{F}(\mathrm{build}_n)"'] \\
A \arrow[uu, "\exists\, g"] \arrow[r, "f"'] & \Sigma_{n:\mathbb{N}}\mathrm{F}^{n+1}(\mathrm{F}^{\infty} + A) \arrow[u, "\Sigma_{n:\mathbb{N}}\mathrm{F}^{n+1}(\mathrm{id} + g)"']
\end{tikzcd}
$$

Rules for coList

- **Formation Rule**

$$\mathrm{coList} : \mathrm{Set}$$

- **Elimination Rule**

$$\mathrm{case} : \mathrm{coList} \longrightarrow \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, \mathrm{coList})$$

- **Introduction Rule**

$$\frac{A : \mathrm{Set} \qquad f : A \longrightarrow \mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, A)}{\mathrm{intro}\ A\ f : A \longrightarrow \mathrm{coList}}$$

- **Equality Rule**

$$\mathrm{case}\ (\mathrm{intro}\ A\ f\ a) = (\mathrm{nil}' + \mathrm{cons}'(\mathbb{N}, \mathrm{intro}\ A\ f))\ (f\ a)$$

Rules for coList

- $\mathrm{intro}\ A\ f\ a = \mathrm{intro}\ A'\ f'\ a'$
  if $A = A'$ and $f = f'$ and $a = a'$.

Suggested Agda Syntax

-

$$
\begin{array}{l}
\text{mutual} \\
\quad \text{coalg coList} : \text{Set where} \\
\quad\quad \text{case} : \text{coList} \to \text{coListShape} \\
\\
\quad \text{data coListShape} : \text{Set where} \\
\quad\quad \begin{array}{lcl} \text{nil}' & : & \text{coListShape} \\ \text{cons}' & : & \mathbb{N} \to \text{coList} \to \text{coListShape} \end{array} \\
\omega : \text{coList} \\
\text{case}\ \omega = \text{cons}'\ 0\ \omega
\end{array}
$$

Getting Close to Codata

- Define (idea by Nils Danielsson):

$$\begin{array}{l} \mathrm{coalg}\ \Box\ (A : \mathrm{Set}) : \mathrm{Set}\ \mathrm{where} \\ \quad \mathrm{case} : \Box\ A \to A \end{array}$$

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

$$\begin{array}{l} \mathrm{coList} : \mathrm{Set} \\ \mathrm{coList} = \Box\ \mathrm{coListShape} \end{array}$$

$$\begin{array}{l} \mathrm{nil} : \mathrm{coList} \\ \mathrm{case}\ \mathrm{nil} = \mathrm{nil}' \end{array}$$

Simplification by Nils Danielsson

- $$\begin{aligned}&\mathrm{cons} : \mathbb{N} \rightarrow \mathrm{coList} \rightarrow \mathrm{coList}\\&\mathrm{case}\ (\mathrm{cons}\ n\ l) = \mathrm{cons}'\ n\ l\end{aligned}$$

- Write
  $$r \sim s$$
  for
  $$\mathrm{case}\ r = \mathrm{case}\ s$$

- Now
  $$\begin{aligned}&\omega : \mathrm{coList}\\&\mathrm{case}\ \omega = \mathrm{cons}'\ n\ \omega\end{aligned}$$
  can be replaced by
  $$\begin{aligned}&\omega : \mathrm{coList}\\&\omega\ \sim\ \mathrm{cons}\ n\ \omega\end{aligned}$$

Subject Reduction Revisited

-

$$
\begin{array}{l}
\text{data } \_ == \_ \; (x : \text{coList}) : \text{coList} \to \text{Set where} \\
\quad \text{refl} : a == a \\
\\
\text{out} : \text{coList} \to \text{coList} \\
\text{out } x \text{ with } (\text{case } x) \\
\cdots \mid \text{nil}' \qquad\qquad = \quad \text{nil} \\
\cdots \mid \text{cons}' \; n \; l \qquad = \quad \text{cons } n \; l \\
\\
\text{lemma} : (l : \text{coList}) \to l == \text{out } l \\
\text{lemma } l \textsf{ with } (\text{case } l) \\
\cdots \quad |\text{nil}' \qquad = \quad \{! \;\; !\} \quad \textsf{goal type = } l == \text{nil} \\
\cdots \quad |\text{cons}' \; n \; l' \qquad = \quad \{! \;\; !\} \quad \textsf{goal type = } l' == \text{cons } n \; l'
\end{array}
$$

- The last two goals are not solvable.

IO using Coalg

$$\mathrm{coalg\ IO\ (C : Set)\ (R : C \to Set) : Set\ where}$$
$$\begin{array}{lll} \quad \mathrm{command} & : & \mathrm{IO\ C\ R \to C} \\ \quad \mathrm{next} & : & (p : \mathrm{IO\ C\ R}) \to \mathrm{R\ (command}\ p) \to \mathrm{IO\ C\ R} \end{array}$$

$$progPasswd : \mathrm{IO\ C\ R}$$
$$\begin{array}{lll} \mathrm{command}\ progPasswd & = & \mathrm{write}\ \text{“Password: ”} \\ \mathrm{next}\ progPasswd\ \_ & = & progRead \end{array}$$

$$progRead : \mathrm{IO\ C\ R}$$
$$\begin{array}{lll} \mathrm{command}\ progRead & = & \mathrm{read} \\ \mathrm{next}\ progRead\ s & = & progCheck\ s \end{array}$$

IO using Coalg

$progCheck : \mathrm{String} \rightarrow \mathrm{IO\ C\ R}$

$progCheck\ s = \mathrm{if}\ (s = \text{“123”})\ \mathrm{then}\ progSuccess\ \mathrm{else}\ progFail$

$progSuccess : \mathrm{IO\ C\ R}$

$$\begin{array}{lcl} \mathrm{command}\ progSuccess & = & \mathrm{terminate} \\ \mathrm{next}\ progSuccess\ () & & \end{array}$$

$progFail : \mathrm{IO\ C\ R}$

$$\begin{array}{lcl} \mathrm{command}\ progFail & = & \mathrm{write}\ \text{“Wrong Password!!”} \\ \mathrm{next}\ progFail\ \_ & = & progPasswd \end{array}$$

Cell using Coalg

$$
\begin{aligned}
&\mathrm{coalg}\ \mathrm{Cell} : \mathrm{Set}\ \mathrm{where}\\
&\qquad \mathrm{set} \quad : \quad \mathrm{Cell} \to \mathbb{N} \to \mathrm{Cell}\\
&\qquad \mathrm{get} \quad : \quad \mathrm{Cell} \to \mathbb{N} \times \mathrm{Cell}
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{cell} \ : \mathbb{N} \to \mathrm{Cell}\\
&\mathrm{set}\ (\mathrm{cell}\ n)\ m \quad = \quad \mathrm{cell}\ m\\
&\mathrm{get}\ (\mathrm{cell}\ n) \qquad = \quad \langle n, \mathrm{cell}\ n\rangle
\end{aligned}
$$

4. Proofs by Corecursion

- Let a transition system be given by

$$\begin{array}{lcl} \mathrm{Vertex} & : & \mathrm{Set} \\ \mathrm{Edge} & : & \mathrm{Vertex} \longrightarrow \mathrm{Set} \\ \mathrm{target} & : & (l : \mathrm{Vertex}) \longrightarrow \mathrm{Edge}\ l \longrightarrow \mathrm{Vertex} \end{array}$$

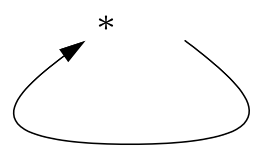
- Let

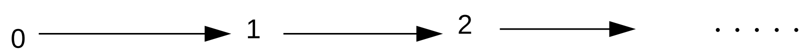
$$(\mathrm{Vertex}_1, \mathrm{Edge}_1, \mathrm{target}_1) = (\{*\}, \lambda\, x.\{*\}, \lambda\, x, y.*)$$

and

$$(\mathrm{Vertex}_2, \mathrm{Edge}_2, \mathrm{target}_2) = (\mathbb{N}, \lambda\, x.\{*\}, \lambda\, x, y.\mathrm{S}\ x)$$

$$
\begin{aligned}
(\mathrm{Vertex}_1, \mathrm{Edge}_1, \mathrm{target}_1) &= (\{*\}, \lambda\, x.\{*\}, \lambda\, x, y.*) \\
(\mathrm{Vertex}_2, \mathrm{Edge}_2, \mathrm{target}_2) &= (\mathbb{N}, \lambda\, x.\{*\}, \lambda\, x, y.\mathrm{S}\; x)
\end{aligned}
$$





Bisimulation

- Define

$$
\begin{array}{l}
\text{coalg Bisim} : (n_1 : \text{Vertex}_1) \to (n_2 : \text{Vertex}_2) \to \text{Set where} \\
\quad \text{toEdge}_2 : (b_1 : \text{Edge}_1\ n_1) \to \text{Edge}_2\ n_2 \\
\quad \text{correctVertex}_2 : (b_1 : \text{Edge}_1\ n_1) \to \text{Bisim} \\
\qquad\qquad\qquad\qquad\qquad\qquad (\text{target}_1\ n_1\ b_1) \\
\qquad\qquad\qquad\qquad\qquad\qquad (\text{target}_2\ n_2\ (\text{toEdge}_2\ b_1)) \\
\quad \text{toEdge}_1 : (b_2 : \text{Edge}_2\ n_2) \to \text{Edge}_1\ n_1 \\
\quad \text{correctVertex}_1 : (b_2 : \text{Edge}_2\ n_2) \to \text{Bisim} \\
\qquad\qquad\qquad\qquad\qquad\qquad (\text{target}_1\ n_1\ (\text{toEdge}_1\ b_2)) \\
\qquad\qquad\qquad\qquad\qquad\qquad (\text{target}_2\ n_2\ b_2)
\end{array}
$$

Proof by Corecursion

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\mathrm{lemma} : (v_1 : \mathrm{Vertex}_1) \longrightarrow (v_2 : \mathrm{Vertex}_2) \longrightarrow \mathrm{Bisim}\ v_1\ v_2} \\
\mathrm{toEdge}_2\ (\mathrm{lemma}\ v_1\ v_2)\ b_1 & = & * \\
\mathrm{correctVertex}_2\ (\mathrm{lemma}\ v_1\ v_2)\ b_1 & = & \mathrm{lemma}\ *\ (\mathrm{S}\ v_2) \\
\mathrm{toEdge}_1\ (\mathrm{lemma}\ v_1\ v_2)\ b_2 & = & * \\
\mathrm{correctVertex}_1\ (\mathrm{lemma}\ v_1\ v_2)\ b_2 & = & \mathrm{lemma}\ *\ (\mathrm{S}\ v_2)
\end{array}
$$

Conclusion

- Coalgebras should be the primary concept, not codata.
- But a good idea to find good abbreviations in order to get close to codata, but these should only be abbreviations.
- Elements of coalgebras represent infinite objects, but are not infinite objects themselves.
- Intensional equality between elements of coalgebras.
- Proofs by corecursion now possible.