Guarded Recursion in Dependent Type Theory

Anton Setzer

1. Interactive Programs.
2. Theory of Coalgebras.
3. Guarded Recursion.
4. Alternative Syntax?

1. Interactive Programs

- Formalisation of Interactive Programs in Dependent Type Theory and reasoning about their correctness.
- Interface for (non-state dependent) interactive program given by
  - a set of commands $\mathrm{C} : \mathrm{Set}$,
  - a set of responses depending on commands $\mathrm{R} : \mathrm{C} \to \mathrm{Set}$.

Examples of Commands/Responses

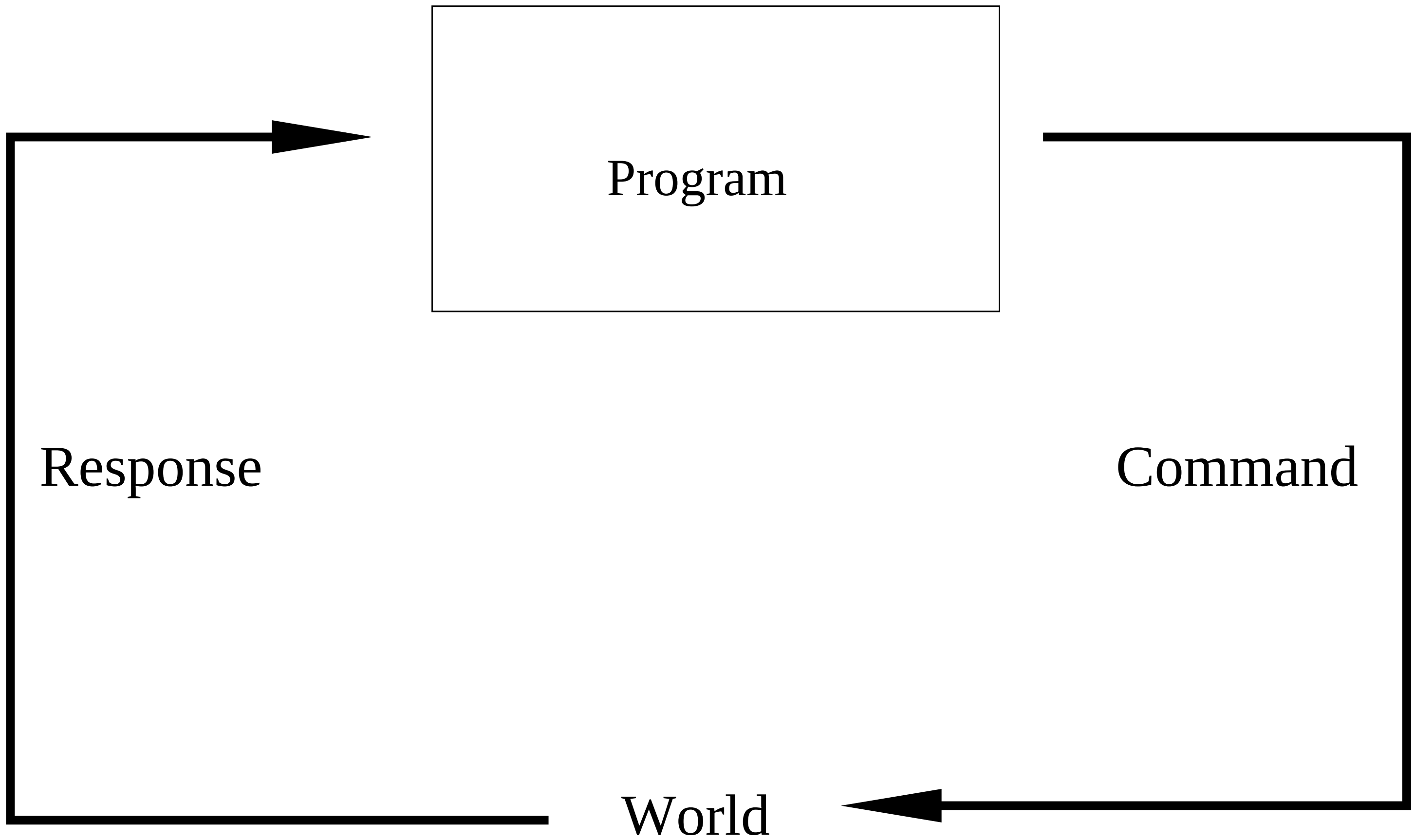
- $\mathrm{writestring}\ (s : \mathrm{String}) : \mathrm{C}$
  - “write string to console”.

  $\mathrm{R}\ (\mathrm{writestring}\ s) = \{*\}$.
  - “empty response”.
- $\mathrm{readtemperature} : \mathrm{C}$
  - “read temperature from a sensor”.

  $\mathrm{R}\ \mathrm{readtemperature} = \mathrm{Temperature}$.

Interactive Program

An interactive program is given by a loop



$\mathrm{IO}\ A$

- In a monadic version we allow as well termination, returning an element $a : A$.
- We obtain $\mathrm{IO} : \mathrm{Set} \to \mathrm{Set}$.
- Then we have that elements of $\mathrm{IO}\ A$ are of the form

$$\mathrm{return}\ (a : A)$$

or

$$\mathrm{do}\ (c : \mathrm{C})\ (\mathit{next} : \mathrm{R}\ c \to \mathrm{IO}\ A)\quad .$$

-

$$\mathrm{IO}\ A \neq \mathrm{data}\ \mathrm{return}\ (a : A) \mid \mathrm{do}\ (c : \mathrm{C})\ (\mathit{next} : \mathrm{R}\ c \to \mathrm{IO}\ A)$$

since interactive programs might run for ever.

Codata

- Instead

$$
\begin{aligned}
\mathrm{IO}\ A = \mathrm{codata}\ \mathrm{return}\ (a : A) & \\
& \mid \mathrm{do}\ (c : \mathrm{C})\ (\mathit{next} : \mathrm{R}\ c \longrightarrow \mathrm{IO}\ A)
\end{aligned}
$$

Execution of $p : \mathrm{IO}\ A$

- Execution $p : \mathrm{IO}\ A$ means iteratively
  - reducing p to weak head normalform
    - if result is
      $$\mathrm{return}\ a$$
      program stops returns a,
    - if result is
      $$\mathrm{do}\ c\ \mathit{next}$$
      command c is executed in real world, if response is $r : \mathrm{R}\ c$, execution continues with executing $\mathit{next}\ r : \mathrm{IO}\ A$.

Execution of $p : \mathrm{IO}\ A$

- Execution of p is not the same as normalisation.
- Differently from Haskell order between commands and responses is guaranteed (even if we have lazy evaluation).
- Direct reasoning about elements of $\mathrm{IO}\ A$ possible.
- Above very generic. For writing programs one can have more userfriendly versions possible like

$$\begin{aligned}\mathrm{IO}\ A = \mathrm{codata}\ \mathrm{return}\ (a : A) & \\ & \mid \mathrm{writestring}\ (s : \mathrm{String})\ (\mathit{next} : \mathrm{IO}\ A) \\ & \mid \mathrm{readtemperature}\ (\mathit{next} : \mathrm{Temperature} \rightarrow \mathrm{IO}\ A)\end{aligned}$$

2. Coalgebras

- We consider stritly positive functors $F : \mathrm{Set} \to \mathrm{Set}$.
  - $\lambda X.A$ is strictly positive.
  - $\lambda X.X$ is strictly positive.
  - If $A : \mathrm{Set}$ and F_a are strictly positive ($a : A$), so are
    - $\lambda X.\Sigma a : A.F_a\ X$,
    - $\lambda X.\Pi a : A.F_a\ X$.
- Precise formalisation possible as a universe of operators (which is a type).
- Morphism part $F\ f : F\ A \to F\ B$ for $f : A \to B$ can be defined.
- Closure under F^∞, F^* possible (not considered here).
- Strictly positive functors closed under $+$:
  $\lambda X.F_0\ X + F_1\ X = \lambda X.\Sigma i : \{0, 1\}.F_i\ X$.

Codata

- As a specific case we consider

$$
\begin{aligned}
F\ X = {} & \mathrm{C}_1\ (\ldots, b : B, \ldots, x : X, \ldots, f : A \to X, \ldots) \\
& + \cdots \\
& + \mathrm{C}_n\ (\ldots)
\end{aligned}
$$

- E.g.

$$
\begin{array}{lcl}
\mathrm{F}_{\mathbb{N}}\ X & = & 0 + \mathrm{S}\ (x : X) \\
\mathrm{F}_{\mathrm{Stream}}\ X & = & \mathrm{cons}\ (n : \mathbb{N})\ (x : X)
\end{array}
$$

Coalgebras

- F_0^∞ = weakly final coalgebra of F.
  - In case of $\mathrm{F}_{\mathbb{N}}\ X = 0 + \mathrm{S}\ (x : X)$ we have

$$\begin{aligned}\mathrm{F}_{\mathbb{N},0}^{\infty} &= \mathrm{codata}\ 0 \mid \mathrm{S}\ (n : \mathrm{F}_{\mathbb{N},0}) \\ &= \mathbb{N}^{\infty}\end{aligned}$$

  - In case of $\mathrm{F}_{\mathrm{Stream}}\ X = \mathrm{cons}\ (n : \mathbb{N})\ (x : X)$ we have

$$\begin{aligned}\mathrm{F}_{\mathrm{Stream},0}^{\infty} &= \mathrm{codata}\ \mathrm{cons}\ (n : \mathbb{N})\ (s : \mathrm{F}_{\mathrm{Stream},0}^{\infty}) \\ &= \mathrm{Stream} \quad .\end{aligned}$$

Monadic Version of F_0^∞

- $F^\infty\ X = (F^X)_0^\infty$ where
  $F^X\ Y = \mathrm{return}\ (x : X) + \mathrm{continue}\ (x : F\ Y)$.
- E.g. (essentially)

$$
\begin{array}{rcl}
F^\infty_{\mathrm{Stream}}\ X & = & \mathrm{codata}\ \mathrm{return}\ (x : X) \\
 & & \qquad\qquad \mid \mathrm{cons}\ (n : \mathbb{N})\ (s : F^\infty_{\mathrm{Stream}}\ X) \\
F^\infty_{\mathbb{N}}\ X & = & \mathrm{codata}\ \mathrm{return}\ (x : X) \\
 & & \qquad\qquad \mid 0 \\
 & & \qquad\qquad \mid \mathrm{S}\ (x : F^\infty_{\mathbb{N}}\ X)
\end{array}
$$

Coalgebras Categorically

- We have $\mathrm{F}_0^\infty : \mathrm{Set}$,
- $\mathrm{elim} : \mathrm{F}_0^\infty \to \mathrm{F}\ \mathrm{F}_0^\infty$,
- s.t. whenever we have $f : A \to \mathrm{F}\ A$ there exists a $g : A \to \mathrm{F}_0^\infty$ s.t.

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "g"'] & \mathrm{F}\ A \arrow[d, "\mathrm{F}\ g"] \\ \mathrm{F}_0^\infty \arrow[r, "\mathrm{elim}"'] & \mathrm{F}\ \mathrm{F}_0^\infty \end{tikzcd}$$

Formation and Elimination Rules

- **Formation Rule for F_0^∞**

$$\mathrm{F}_0^\infty : \mathrm{Set}$$

- **Elimination Rule for F_0^∞**

$$\frac{a : \mathrm{F}_0^\infty}{\mathrm{elim}\ a : \mathrm{F}\ \mathrm{F}_0^\infty}$$

Special Cases

1. $$\mathbb{N}^\infty : \mathrm{Set}$$

$$\frac{n : \mathbb{N}^\infty}{\mathrm{elim}\ n : 0 + \mathrm{S}\ (n : \mathbb{N}^\infty)}$$

2. $$\mathrm{Stream} : \mathrm{Set}$$

$$\frac{l : \mathrm{Stream}}{\mathrm{elim}\ l : \mathrm{cons}\ (n : \mathbb{N})\ (s : \mathrm{Stream})}$$

Agda

- Formation Rules corresponds to formation of

$$A : \mathrm{Set} = \mathrm{codata}\, \mathrm{C}_1\ (\cdots) \mid \cdots \mid \mathrm{C}_k\ (\cdots)$$

- Elimination rule corresponds to possibility of case distinction.

Intro-/Equality Rules

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "\mathrm{intro}\ A\ f"'] & \mathrm{F}\ A \arrow[d, "\mathrm{F}\ (\mathrm{intro}\ A\ f)"] \\
\mathrm{F}_0^\infty \arrow[r, "\mathrm{elim}"'] & \mathrm{F}\ \mathrm{F}_0^\infty
\end{tikzcd}$$

- **Introduction Rule**

$$\frac{A : \mathrm{Set} \qquad f : A \to \mathrm{F}\ A}{\mathrm{intro}\ A\ f : A \to \mathrm{F}_0^\infty}$$

- **Equality Rule**

$$\mathrm{elim}\ (\mathrm{intro}\ A\ f\ a) = F\ (\underbrace{\mathrm{intro}\ A\ f}_{:A \to \mathrm{F}_0^\infty})(\underbrace{f\ a}_{:\mathrm{F}\ A}) : \mathrm{F}\ \mathrm{F}_0^\infty$$

3. Guarded Recursion

- Let

$$\begin{aligned} F\ X = \mathrm{C}_1\ (\ldots, b : B, \ldots, x : X, \ldots, h : E \to X, \ldots) \\ + \cdots \\ + \mathrm{C}_n\ (\ldots, b : B', \ldots, x : X, \ldots, h : E' \to X, \ldots) \end{aligned}$$

- $f : A \to \mathrm{F}\ A$ means $f\ a$ is of the form $\mathrm{C}_i\ (\ldots, b : B, \ldots, a : A, \ldots, h : E \to A, \ldots)$.
- Let $g = \mathrm{intro}\ A\ f : A \to \mathrm{F}^{\infty}$.
- Then the equality rules expresses

$$\mathrm{elim}\ (g\ a) = (F\ g)\ (f\ a)$$

Guarded Recursion

$f\ a$ of the form $\mathrm{C}_i\ (\ldots, b : B, \ldots, a : A, \ldots, h : E \to A, \ldots)$.

- Then $(F\ g)\ (f\ a)$ is

$$\mathrm{C}_i\ (\ldots, b : B, \ldots, g\ a, \ldots, g \circ h, \ldots)$$

- $\mathrm{elim}\ (g\ a) = (F\ g)\ (f\ a)$ means that

$$\mathrm{elim}\ (g\ a) = \mathrm{C}_i\ (\ldots, b : B, \ldots, g\ a, \ldots, g \circ h, \ldots)$$

- So the introduction rule means that we can define $g : A \to \mathrm{F}_0^\infty$ s.t.

$$\mathrm{elim}\ (g\ a) = \mathrm{C}_i\ (\ldots, b : B, \ldots, g\ a, \ldots, g \circ h, \ldots)$$

Generalised Intro/Equality Rule

- **Generalised Introduction Rule**

$$\frac{A : \mathrm{Set} \qquad f : A \longrightarrow \mathrm{F}\ (\mathrm{F}^{\infty}\ (A + \mathrm{F}_0^{\infty}))}{\mathrm{intro}'\ A\ f : A \longrightarrow \mathrm{F}_0^{\infty}}$$

- **Generalised Equality Rule**

$$\begin{aligned}
&\mathrm{elim}'\ (\mathrm{intro}'\ A\ f\ a)\\
&= F\ (g \circ \underbrace{\mathrm{F}^{\infty}\ (\underbrace{[\mathrm{intro}'\ A\ f, \lambda x.x]}_{(A+\mathrm{F}_0^{\infty})\to\mathrm{F}_0^{\infty}})}_{\mathrm{F}^{\infty}\ (A+\mathrm{F}_0^{\infty})\to\mathrm{F}^{\infty}\ \mathrm{F}_0^{\infty}})\ (\underbrace{f\ a}_{:\mathrm{F}\ (\mathrm{F}^{\infty}\ (A+\mathrm{F}_0^{\infty}))})
\end{aligned}$$

where $g : \mathrm{F}^{\infty}\ \mathrm{F}_0^{\infty} \longrightarrow \mathrm{F}_0^{\infty}$.

- Can be essentially reduced to the above.

Generalised Guarded Recursion

- In case

$$F\ X = \mathrm{C}_1\ (\ldots, b : B, \ldots, x : X, \ldots) \\ +\ \cdots \\ +\ \mathrm{C}_n\ (\ldots, b : B', \ldots, x : X, \ldots)$$

the rules mean:

- we can define $f : A \to \mathrm{F}_0^\infty$ s.t.

$$\mathrm{elim}\ (f\ a) = \mathrm{C}_i\ (\ldots, b, \ldots, \mathrm{C}_j\ (\ldots, \mathrm{C}_k\ (\ldots, t, \ldots), \ldots), \ldots)\ \ldots)\ .$$

where $t : \mathrm{F}_0^\infty$ or $t = f\ a'$.

Agda

- So the introduction/equality rules mean that if

$$A = \mathrm{codata}\ \mathrm{C}_1\ (\ldots) \mid \cdots \mid \mathrm{C}_n\ (\ldots)$$

  we can define

$$\begin{aligned} & g : B \to A \\ & \quad \text{where} \\ & \quad \mathrm{elim}\ (g\ a) = t \end{aligned}$$

  where t is a guarded recursion pattern.
- Dependent version possible as well.

Constructors

$A = \mathrm{codata}\ \mathrm{C}_1\ (\ldots) \mid \cdots \mid \mathrm{C}_n\ (\ldots)$

- A convenient syntactic sugar would be to have
  $\mathrm{C}_i\ a_0 \cdots\ a_n : A$

  which is the b given by

  $$\begin{array}{l} b : A \\ \quad \mathrm{where} \\ \quad \mathrm{elim}\ b = \mathrm{C}_i\ a_0 \cdots a_n \end{array}$$

Example

- $\text{Stream} = \text{codata cons } (n : \mathbb{N})\ (s : \text{Stream})$.
- Define

$$f : \mathbb{N} \to \text{Stream}$$
$$\text{where}$$
$$\text{elim } (f\ n) = \text{cons } n\ (f\ (n+1))$$

- Define

$$g : \mathbb{N} \to \text{Stream}$$
$$\text{where}$$
$$\text{elim } (g\ n) = \text{cons } n\ (\text{cons } (n+1)\ (g\ (n+2)))$$

Bisimulation

- $\mathrm{Bisim} : \mathrm{Stream} \to \mathrm{Stream} \to \mathrm{Set}$
- $\mathrm{Bisim}\ s\ s' =$
  $\mathrm{case\ elim}\ s\ \mathrm{of}$
  $\quad (\mathrm{cons}\ n\ t)$
  $\quad \longrightarrow \mathrm{case\ elim}\ s'\ \mathrm{of}$
  $\qquad (\mathrm{cons}\ n'\ t')$
  $\qquad \longrightarrow \mathrm{codata\ bisim}\ (p : n == n')\ (q : \mathrm{Bisim}\ t\ t')$

Proof by Coninduction

$$
\begin{array}{l}
\text{mutual} \\
\quad \text{lem}_1 : (n : \mathbb{N}) \to \text{Bisim } (f\ n)\ (g\ n) \\
\qquad \text{where} \\
\qquad \text{elim } (\text{lem}_1\ n) = \text{bisim } (\text{refl } n)\ (\text{lem}_2\ (n+1)) \\
\qquad\qquad \{-\text{note that} \\
\qquad\qquad\quad \text{elim } (f\ n) = \text{cons } n\ (f\ (n+1)) \\
\qquad\qquad\quad \text{elim } (g\ n) = \text{cons } n\ (\text{cons } (n+1)\ (g\ (n+2)))\ -\} \\
\quad \text{lem}_2 : (n : \mathbb{N}) \to \text{Bisim } (f\ n)\ (\text{cons } n\ (g\ (n+1))) \\
\qquad \text{where} \\
\qquad \text{elim } (\text{lem}_2\ n) = \text{bisim } (\text{refl } n)\ (\text{lem}_1\ n) \\
\qquad\qquad \{-\text{note that} \\
\qquad\qquad\quad \text{elim } (f\ n) = \text{cons } n\ (f\ (n+1)) \\
\qquad\qquad\quad \text{elim } (\text{cons } n\ (g\ (n+1))) = \text{cons } n\ (g\ (n+1))\ -\}
\end{array}
$$

4. Alternative Syntax?

- Algebraic data types are given by their introduction rules.
- Coalgebraic types are given by their elimination rule.
- Maybe we should have instead of the above

$$
\begin{array}{lcl}
\mathbb{N}^\infty & = & \mathrm{coalg} \\
 & & \quad \mathrm{elim}_{\mathbb{N}} \quad : \quad \mathrm{data}\ 0 \mid \mathrm{S}\ (n : \mathbb{N}^\infty) \\
\mathrm{Stream} & = & \mathrm{coalg} \\
 & & \quad \mathrm{elim}_{\mathrm{Stream}} \quad : \quad \mathrm{record}\ (n : \mathbb{N})\ (s : \mathrm{Stream})
\end{array}
$$

or

$$
\begin{array}{lcl}
\mathrm{Stream} & = & \mathrm{coalg} \\
 & & \quad \mathrm{head} \quad : \quad \mathbb{N} \\
 & & \quad \mathrm{tail} \quad : \quad \mathrm{Stream}
\end{array}
$$

Alternative Syntax?

- Then we would have,
  - if $n : \mathbb{N}$ then

$$n.\mathrm{elim}_{\mathbb{N}} : \mathrm{data}\ 0 \mid \mathrm{S}\ (n : \mathbb{N}^{\infty})$$

  - if $s : \mathrm{Stream}$ then

$$n.\mathrm{elim}_{\mathrm{Stream}} : \mathrm{record}\ (n : \mathbb{N})\ (s : \mathrm{Stream})$$

- And we have

$$\begin{array}{l} f : \mathbb{N} \longrightarrow \mathbb{N}^{\infty} \\ \quad \mathrm{where} \\ \quad (f\ n).\mathrm{elim} = \mathrm{S}\ (f\ n) \end{array}$$

Alternative Syntax?

- Or we have if

$$
\begin{array}{lll}
\mathrm{Stream} & = & \mathrm{coalg} \\
 & & \begin{array}{lll} \mathrm{head} & : & \mathbb{N} \\ \mathrm{tail} & : & \mathrm{Stream} \end{array}
\end{array}
$$

then we can define

$$
\begin{array}{l}
f : \mathbb{N} \rightarrow \mathrm{Stream} \\
\quad \mathrm{where} \\
\quad (f\ n).\mathrm{head} = n \\
\quad (f\ n).\mathrm{tail} = f\ (n+1)
\end{array}
$$

Alternative Syntax?

- $$\begin{array}{lcl} \mathrm{Bisim}\ (s, s' : \mathrm{Stream}) & = & \mathrm{coalg} \\ & & \quad \begin{array}{lcl} \mathrm{head}_= & : & s.\mathrm{head} == s'.\mathrm{head} \\ \mathrm{tail}_= & : & \mathrm{Bisim}\ s.\mathrm{tail}\ s'.\mathrm{tail} \end{array} \end{array}$$

Alternative Syntax?

- Then

$$\mathbb{N}^\infty = \text{codata } 0 \mid \mathrm{S}\ (n : \mathbb{N}^\infty)$$

would be an abbreviation for

$$\begin{array}{ll}
\mathbb{N}^\infty & = \text{coalg} \\
& \qquad \text{case} : \text{data } 0 \mid \mathrm{S}\ (n : \mathbb{N}^\infty) \\
0 & : \mathbb{N}^\infty \\
& \qquad \text{where} \\
& \qquad 0.\text{case} = 0 \\
\mathrm{S} & : \mathbb{N}^\infty \to \mathbb{N}^\infty \\
& \quad \text{where} \\
& \quad (\mathrm{S}\ n).\text{case} = \mathrm{S}\ n
\end{array}$$

Conclusion

- $\mathrm{IO}\ A$ is a special case of codata.
- Convenient syntax for codata.
- “data” types are determined by their introduction rules
  - elimination rules are “derived” and impredicative.
- “codata” types are determined by their elimination rules
  - introduction rules are “derived” and impredicative.
- Guarded recursion not to be read as an equality creating infinite terms.
  - $\mathrm{elim}\ n = \mathrm{S}\ n$ rather than $n = \mathrm{S}\ n$.
- Bisimulation dependent codata type.
- Proofs of bisimulation can be done by guarded recursion.