# Impredicative Encodings of Inductive and Coinductive Types

Steven Bronsveld, Herman Geuvers, Niels van der Weide

## Impredicative Encodings

- ► Impredicative encodings allow us to reduce inductive types to elementary type formers: ∏, →
- This is how one would implement them in Rocq in the past
- Only suitable in impredicative settings

## Impredicative Encodings

- ► Impredicative encodings allow us to reduce inductive types to elementary type formers: ∏, →
- This is how one would implement them in Rocq in the past
- Only suitable in impredicative settings

Impredicativity: we have an impredicative universe  ${\cal U}$  closed under = and  $\sum$  and the following rule

$$\frac{\Gamma \vdash A \operatorname{Type} \quad \Gamma, x : A \vdash B x : \mathcal{U}}{\Gamma \vdash \prod (x : A), B x : \mathcal{U}}$$

Let *E* be a type. Define List<sup>\*</sup> :  $\mathcal{U}$  as follows.

$$\mathsf{List}^* = \prod(X : \mathcal{U}), X \to (E \to X \to X) \to X$$

Let *E* be a type. Define List<sup>\*</sup> :  $\mathcal{U}$  as follows.

$$\mathsf{List}^* = \prod(X:\mathcal{U}), X \to (E \to X \to X) \to X$$

We can define:

 $\mathsf{nil}^*:\mathsf{List}^*$  $\mathsf{nil}^* = \lambda(X:\mathcal{U})(n:X)(c:E \to X \to X), n$ 

Let *E* be a type. Define List<sup>\*</sup> :  $\mathcal{U}$  as follows.

$$\mathsf{List}^* = \prod(X:\mathcal{U}), X \to (E \to X \to X) \to X$$

We can define:

$$\begin{split} \mathsf{nil}^* &: \mathsf{List}^* \\ \mathsf{nil}^* &= \lambda(X:\mathcal{U})(n:X)(c:E \to X \to X), n \\ &\quad \mathsf{cons}^* : E \to \mathsf{List}^* \to \mathsf{List}^* \\ \mathsf{cons}^* \; e \; l &= \lambda(X:\mathcal{U})(n:X)(c:E \to X \to X), c \; e \; l \end{split}$$

Let *E* be a type. Define List<sup>\*</sup> :  $\mathcal{U}$  as follows.

$$\mathsf{List}^* = \prod(X:\mathcal{U}), X \to (E \to X \to X) \to X$$

We can define:

nil\* : List\*  $\operatorname{nil}^* = \lambda(X : \mathcal{U})(n : X)(c : E \to X \to X), n$  $cons^* : E \rightarrow List^* \rightarrow List^*$  $cons^* e I = \lambda(X : U)(n : X)(c : E \to X \to X), c e I$  $\operatorname{rec}_{\operatorname{List}^*}$ :  $(X : \mathcal{U}), X \to (E \to X \to X) \to \operatorname{List}^* \to X$  $\operatorname{rec}_{\operatorname{Iist}^*} X n c = \lambda(I : \operatorname{List}^*), I X n c$ 

- What do we want of inductive types? Induction principles!
- For List\*, we can prove the recursion principle with the expected β-rules
- However, induction is not derivable<sup>1</sup>

List\* is not an initial algebra, uniqueness does not hold in general.

<sup>&</sup>lt;sup>1</sup>Geuvers, "Induction is not derivable in second order dependent type theory"

Awodey, Frey, and Speight: don't worry, we can fix this <sup>2</sup>

- Intuition: the type List\* has "too many inhabitants"
- Define a predicate Lim<sub>List</sub> on List<sup>\*</sup> (next slide)
- Define List to be  $\sum (I : \text{List}^*)$ ,  $\lim_{\text{List}} I$

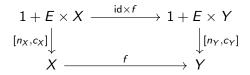
 $<sup>^2\</sup>mathsf{Awodey},$  Frey, Speight, "Impredicative encodings of (higher) inductive types"

Awodey, Frey, and Speight: don't worry, we can fix this <sup>2</sup>

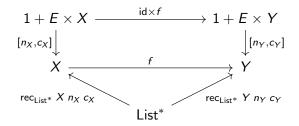
- Intuition: the type List\* has "too many inhabitants"
- Define a predicate Lim<sub>List</sub> on List<sup>\*</sup> (next slide)
- Define List to be  $\sum (I : \text{List}^*)$ ,  $\lim_{\text{List}} I$
- One can prove that List is an initial algebra
- Initial algebra semantics: List satisfies induction

 $<sup>^{2}\</sup>mathsf{Awodey},$  Frey, Speight, "Impredicative encodings of (higher) inductive types"

To define Lim<sub>List</sub>: Suppose we have a commuting square.



To define Lim<sub>List</sub>: Suppose we have a commuting square.



Then the bottom triangle must commute.

We say that  $I : List^*$  satisfies  $Lim_{List}$  if for all

- ▶ X : U together with  $n_X : X$ ,  $c_X : E \to X \to X$
- ▶ Y : U together with  $n_Y : Y$ ,  $c_Y : E \to Y \to Y$

We say that I : List\* satisfies Lim<sub>List</sub> if for all

- ▶ X : U together with  $n_X : X$ ,  $c_X : E \to X \to X$
- ▶ Y : U together with  $n_Y : Y, c_Y : E \to Y \to Y$

$$\blacktriangleright f: X \to Y$$

$$\blacktriangleright p_n : f n_X = n_Y$$

• 
$$p_c: \prod (e:E)(x:X), f(c_X e X) = c_Y e(f x)$$

We say that  $I : List^*$  satisfies  $Lim_{List}$  if for all

- ▶ X : U together with  $n_X : X, c_X : E \to X \to X$
- ▶ Y : U together with  $n_Y : Y$ ,  $c_Y : E \to Y \to Y$

• 
$$f: X \to Y$$

we have

$$f\left(\operatorname{rec}_{\operatorname{List}^{*}}X n_{X} c_{X} l\right) = \operatorname{rec}_{\operatorname{List}^{*}}Y n_{Y} c_{Y} l$$

# Other Encodings

Awodey, Frey, and Speight considered

- sum types
- algebras for a functor on sets (i.e., types for which there's at most one proof that x = y)
- natural numbers
- the circle

They worked in a setting without uniqueness of identity proofs

<sup>3</sup>Echeveste, "Alternative impredicative encodings of inductive types" <sup>4</sup>https://homotopytypetheory.org/2018/11/26/ impredicative-encodings-part-3/

# Other Encodings

Awodey, Frey, and Speight considered

sum types

- algebras for a functor on sets (i.e., types for which there's at most one proof that x = y)
- natural numbers
- the circle

They worked in a setting without uniqueness of identity proofs Note: one can get rid of the truncation assumption  $^{3\ 4}$ 

<sup>3</sup>Echeveste, "Alternative impredicative encodings of inductive types" <sup>4</sup>https://homotopytypetheory.org/2018/11/26/ impredicative-encodings-part-3/

### This work: coinductive types

We look at the dualization

- define coinductive types using impredicative encodings
- prove suitable coinduction principles, i.e., bisimulation corresponds to equality

## This work: coinductive types

We look at the dualization

- define coinductive types using impredicative encodings
- prove suitable coinduction principles, i.e., bisimulation corresponds to equality

This talk:

- we demonstrate the method for streams
- we discuss how to extend it to M-types

#### Introduction

### Streams via Imprecative Encodings

Extension to M-Types

Conclusion

#### Introduction

### Streams via Imprecative Encodings

Extension to M-Types

Conclusion

### Main Idea

Recall:

$$\mathsf{List}^* = \prod(X : \mathcal{U}), X \to (E \to X \to X) \to X$$
$$\mathsf{List} = \sum(I : \mathsf{List}^*), \mathsf{Lim}_{\mathsf{List}} I$$

To dualize this construction:

- To dualize  $\prod$ , we use **existential types**
- To dualize the subtype: we use quotient types

### Main Idea

Recall:

$$\mathsf{List}^* = \prod(X : \mathcal{U}), X \to (E \to X \to X) \to X$$
$$\mathsf{List} = \sum(I : \mathsf{List}^*), \mathsf{Lim}_{\mathsf{List}} I$$

To dualize this construction:

- To dualize  $\prod$ , we use **existential types**
- To dualize the subtype: we use **quotient types**

We define existential types and quotient types via impredicative encodings.

## Existential Types

Let  $P:\mathcal{U}\rightarrow\mathcal{U}$  be a type family. Then we have

$$\blacktriangleright \exists (X:\mathcal{U}), PX:\mathcal{U}$$

▶ pack : 
$$\prod(X : U), P X \rightarrow \exists (X : U), P X$$

### Existential Types

Let  $P:\mathcal{U}\rightarrow\mathcal{U}$  be a type family. Then we have

$$\blacktriangleright \exists (X:\mathcal{U}), PX:\mathcal{U}$$

▶ pack : 
$$\prod(X : U), P X \rightarrow \exists (X : U), P X$$

together with a recursion principle:

$$\mathsf{rec}_{\exists} : \prod(Y : \mathcal{U}),$$
$$(\prod(Z : \mathcal{U}), P Z \to Y)$$
$$\to (\exists (X : \mathcal{U}), P X)$$
$$\to Y$$

satisfying the expected  $\beta$ - and  $\eta$ -rules.

### Existential Types

Impredicative encoding: we define  $\exists^*(X : U), P X$  to be

$$\prod(Y:\mathcal{U}), (\prod(Z:\mathcal{U}), (P \ Z \to Y) \to Y) \to Y$$

We define  $Lim_{\exists}$  similarly to  $Lim_{List}$  and

$$\exists (X : U), P X = \sum (x : \exists^* (X : U), P X), \operatorname{Lim}_\exists x$$

## **Encoding Streams**

Let *E* be a type. We define Stream<sup>\*</sup> as follows<sup>5</sup>.

$$\mathsf{Stream}^* = \exists (X:\mathcal{U}), X imes (X o E) imes (X o X)$$

This allows us to define:

- ▶  $hd^*$  : Stream<sup>\*</sup> → E
- $\blacktriangleright tl^*: Stream^* \rightarrow Stream^*$
- ► corec :  $\prod(X : U), (X \to E) \to (X \to X) \to X \to \text{Stream}^*$

 $<sup>^5\</sup>mbox{Geuvers.}$  "The Church-Scott representation of inductive and coinductive data"

### Let's see how to define $tI^*$ : Stream<sup>\*</sup> $\rightarrow$ Stream<sup>\*</sup>.

$$tl^* s = ?$$

where  $?: Stream^*$ 

Let's see how to define  $tI^*$  : Stream<sup>\*</sup>  $\rightarrow$  Stream<sup>\*</sup>.

$$\mathsf{tl}^* s = \mathsf{rec}_\exists \mathsf{Stream}^* ? s$$

where  $?: \prod (Z: U), Z \times (Z \rightarrow E) \times (Z \rightarrow Z) \rightarrow \mathsf{Stream}^*$ 

Let's see how to define  $\mathsf{tl}^*:\mathsf{Stream}^*\to\mathsf{Stream}^*.$ 

$$\mathsf{tl}^* s = \mathsf{rec}_\exists \operatorname{Stream}^* (\lambda Z \, z \, h \, t, ?) \, s$$

where ? : Stream\* Here:

- ► Z : U
- ► z : Z
- ▶  $h: Z \to E$
- $t: Z \to Z$

Let's see how to define tl\* : Stream\*  $\rightarrow$  Stream\*.

$$\mathsf{tl}^* s = \mathsf{rec}_\exists \operatorname{Stream}^* (\lambda Z \, z \, h \, t, \operatorname{pack} Z \, ?) s$$

where  $?: Z \times (Z \rightarrow E) \times (Z \rightarrow Z)$ Here:

- ► Z : U
- ► z : Z
- ►  $h: Z \to E$
- $t: Z \to Z$

Let's see how to define  $\mathsf{tl}^*:\mathsf{Stream}^*\to\mathsf{Stream}^*.$ 

$$\mathsf{tl}^*\,s = \mathsf{rec}_\exists\,\mathsf{Stream}^*\,ig(\lambda Z\,z\,h\,t,\mathsf{pack}\,Z\,ig\langle t\,z,h,tig
angleig)\,s$$

Here:

- ► Z : U
   ► z : Z
- ►  $h: Z \to E$
- ►  $t: Z \to Z$

Let's see how to define  $\mathsf{tl}^*:\mathsf{Stream}^*\to\mathsf{Stream}^*.$ 

$$\mathsf{tl}^* \, s = \mathsf{rec}_\exists \, \mathsf{Stream}^* \left( \lambda Z \, z \, h \, t, \mathsf{pack} \, Z \, \langle t \, z, h, t 
angle 
ight) s$$

Here:

- ► Z : U
   ► z : Z
- ►  $h: Z \to E$
- ►  $t: Z \to Z$

## Fixing the Impredicative Encoding for Streams

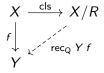
- Just like for lists, we cannot prove a suitable coinduction principle for Stream\*.
- Fix for lists: take a subtype
- Fix for streams: take a **quotient**

## Quotient Types

Using impredicative encodings, we construct quotient types Let X : U and let  $R : X \to X \to U$  be a relation. Then we have

- ▶ a type X/R : U
- ▶ a function cls :  $X \to X/R$

For all Y : U and  $f : X \to Y$  that respects R, there is a unique rec<sub>Q</sub> Y f making the following diagram commute



## Quotient Types

The starting point is the following type:

$$X/^*R = \prod (Z:\mathcal{U})(f:X \to Z), \text{resp } f \ R \to Z$$

Here resp f R says that f respects R.

# Quotient Types

The starting point is the following type:

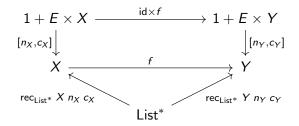
$$X/^*R = \prod (Z:\mathcal{U})(f:X \to Z), \text{resp } f \ R \to Z$$

Here resp f R says that f respects R. We define  $Lim_Q$  similarly to  $Lim_{List}$  and

$$X/R = \sum (x : X/^*R), \operatorname{Lim}_{\mathsf{Q}} x$$

Recall: Fixing Impredicative Encodings for Lists

To define Lim<sub>List</sub>: Suppose we have a commuting square.

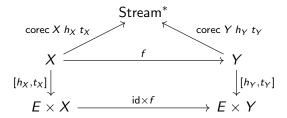


Then the bottom triangle must commute.

Suppose we have a commuting square.

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ [h_X, t_X] \downarrow & & \downarrow [h_Y, t_Y] \\ E \times X & \xrightarrow{\operatorname{id} \times f} & E \times Y \end{array}$$

Suppose we have a commuting square.



Then the upper triangle must commute

Given  $\sigma, \tau$  : Stream<sup>\*</sup>, we say  $\sigma \equiv \tau$  if

$$\exists (X : \mathcal{U})(h_X : X \to E)(t_X : X \to X) \ (Y : \mathcal{U})(h_Y : Y \to E)(t_Y : Y \to Y)$$

Given  $\sigma, \tau$  : Stream<sup>\*</sup>, we say  $\sigma \equiv \tau$  if

$$\exists (X : U)(h_X : X \to E)(t_X : X \to X) \ (Y : U)(h_Y : Y \to E)(t_Y : Y \to Y) \ (f : X \to Y)$$

Given  $\sigma,\tau:\mathsf{Stream}^*\text{, we say }\sigma\equiv\tau$  if

$$\exists (X : \mathcal{U})(h_X : X \to E)(t_X : X \to X) (Y : \mathcal{U})(h_Y : Y \to E)(t_Y : Y \to Y) (f : X \to Y) (p_h : \prod(x : X), h_X x = h_Y(f y)) (p_t : \prod(x : X), t_Y (f x) = f (t_X x))$$

Given  $\sigma, \tau$  : Stream<sup>\*</sup>, we say  $\sigma \equiv \tau$  if

$$\exists (X : \mathcal{U})(h_X : X \to E)(t_X : X \to X) (Y : \mathcal{U})(h_Y : Y \to E)(t_Y : Y \to Y) (f : X \to Y) (p_h : \prod(x : X), h_X x = h_Y(f y)) (p_t : \prod(x : X), t_Y (f x) = f (t_X x)) (x : X), \sigma = \operatorname{corec} X h_X t_X x \land \tau = \operatorname{corec} Y h_Y t_Y (f x)$$

Given  $\sigma, \tau$  : Stream<sup>\*</sup>, we say  $\sigma \equiv \tau$  if

$$\exists (X : \mathcal{U})(h_X : X \to E)(t_X : X \to X) (Y : \mathcal{U})(h_Y : Y \to E)(t_Y : Y \to Y) (f : X \to Y) (p_h : \prod(x : X), h_X x = h_Y(f y)) (p_t : \prod(x : X), t_Y (f x) = f (t_X x)) (x : X), \sigma = \operatorname{corec} X h_X t_X x \land \tau = \operatorname{corec} Y h_Y t_Y (f x)$$

Define Stream = Stream<sup>\*</sup>/ $\equiv$ .

#### Introduction

#### Streams via Imprecative Encodings

#### Extension to M-Types

Conclusion

# Extending to M-types

Recall: M-types are final coalgebras for polynomial functors"

$$P(X) = \sum (a : A), B a o X$$

For M-types, we can take the same steps

# Extending to M-types

Recall: M-types are final coalgebras for polynomial functors"

$$P(X) = \sum (a:A), B a \to X$$

- For M-types, we can take the same steps
- ▶ We first *M*<sup>\*</sup>(*A*, *B*) using existential types
- Then we define a relation  $\equiv$  on  $M^*(A, B)$
- We define M(A, B) as the quotient  $M^*(A, B)/\equiv$

The Impredicative Encoding for M-types

Let  $A : \mathcal{U}$  and  $B : A \rightarrow \mathcal{U}$ . Define

$$M^*(A,B) = \exists (X:\mathcal{U}), X imes (X o \sum (a:A), B \ a o X)$$

#### Introduction

#### Streams via Imprecative Encodings

Extension to M-Types

#### Conclusion

# Conclusion

Key points:

- We can use impredicative encodings to define inductive and coinductive types
- For inductive types: use a subtype (Awodey, Frey, Speight)
- Dual for coinductive types: use existential and quotient types
- This method works for M-types