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: $\prod$, $\to$
- 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: $\prod$, $\rightarrow$
- This is how one would implement them in Rocq in the past
- Only suitable in impredicative settings

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

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

Impredicative Encoding of Lists

Let E be a type. Define $\mathsf{List}^* : \mathcal{U}$ as follows.

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

Impredicative Encoding of Lists

Let E be a type. Define $\mathsf{List}^* : \mathcal{U}$ as follows.

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

We can define:

$$\mathsf{nil}^* : \mathsf{List}^*$$

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

Impredicative Encoding of Lists

Let E be a type. Define $\mathsf{List}^* : \mathcal{U}$ as follows.

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

We can define:

$$\mathsf{nil}^* : \mathsf{List}^*$$

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

$$\mathsf{cons}^* : E \to \mathsf{List}^* \to \mathsf{List}^*$$

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

Impredicative Encoding of Lists

Let E be a type. Define $\mathsf{List}^* : \mathcal{U}$ as follows.

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

We can define:

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

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

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

But.....

- 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**[1]

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

[1] Geuvers, "Induction is not derivable in second order dependent type theory"

Fixing Impredicative Encodings

Awodey, Frey, and Speight: don't worry, we can fix this [2]

- Intuition: the type List^* has "too many inhabitants"
- Define a predicate $\mathsf{Lim}_{\mathsf{List}}$ on List^* (next slide)
- Define List to be $\sum(l : \mathsf{List}^*), \mathsf{Lim}_{\mathsf{List}}\ l$

[2] Awodey, Frey, Speight, "Impredicative encodings of (higher) inductive types"

Fixing Impredicative Encodings

Awodey, Frey, and Speight: don't worry, we can fix this [2]

- Intuition: the type List^* has "too many inhabitants"
- Define a predicate $\mathsf{Lim}_{\mathsf{List}}$ on List^* (next slide)
- Define List to be $\sum(l : \mathsf{List}^*), \mathsf{Lim}_{\mathsf{List}}\ l$
- One can prove that List is an initial algebra
- Initial algebra semantics: List satisfies induction

[2] Awodey, Frey, Speight, "Impredicative encodings of (higher) inductive types"

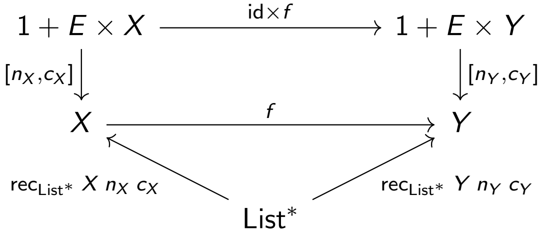
Fixing Impredicative Encodings

To define $\mathrm{Lim}_{\mathrm{List}}$:
Suppose we have a commuting square.

$$
\begin{array}{ccc}
1 + E \times X & \xrightarrow{\mathrm{id} \times f} & 1 + E \times Y \\
{\scriptstyle [n_X, c_X]} \big\downarrow & & \big\downarrow {\scriptstyle [n_Y, c_Y]} \\
X & \xrightarrow{\quad f \quad} & Y
\end{array}
$$

Fixing Impredicative Encodings

To define $\mathrm{Lim}_{\mathrm{List}}$:
Suppose we have a commuting square.



Then the bottom triangle must commute.

Fixing Impredicative Encodings

We say that $l : \mathsf{List}^*$ satisfies $\mathsf{Lim}_{\mathsf{List}}$ if for all

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

Fixing Impredicative Encodings

We say that l : List* satisfies $\mathsf{Lim}_{\mathsf{List}}$ if for all

- $X : \mathcal{U}$ together with $n_X : X$, $c_X : E \to X \to X$
- $Y : \mathcal{U}$ together with $n_Y : Y$, $c_Y : E \to Y \to Y$
- $f : X \to Y$
- $p_n : f\ n_X = n_Y$
- $p_c : \prod(e : E)(x : X), f\ (c_X\ e\ X) = c_Y\ e\ (f\ x)$

Fixing Impredicative Encodings

We say that $l : \text{List}^*$ satisfies Lim_{List} if for all

- $X : \mathcal{U}$ together with $n_X : X$, $c_X : E \to X \to X$
- $Y : \mathcal{U}$ together with $n_Y : Y$, $c_Y : E \to Y \to Y$
- $f : X \to Y$
- $p_n : f\ n_X = n_Y$
- $p_c : \prod(e : E)(x : X), f\ (c_X\ e\ X) = c_Y\ e\ (f\ x)$

we have

$$f\ (\text{rec}_{\text{List}^*}\ X\ n_X\ c_X\ l) = \text{rec}_{\text{List}^*}\ Y\ n_Y\ c_Y\ l$$

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

[3] Echeveste, "Alternative impredicative encodings of inductive types"

[4] `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]

[3] Echeveste, "Alternative impredicative encodings of inductive types"

[4] `https://homotopytypetheory.org/2018/11/26/impredicative-encodings-part-3/`

This work: coinductive types

We look at the dualization

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

This talk: how to define streams using impredicative encodings

Main Idea

Recall:

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

$$\mathsf{List} = \sum(l : \mathsf{List}^*), \mathsf{Lim}_{\mathsf{List}}\ l$$

To dualize this construction:

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

Existential Types

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

- $\exists(X : \mathcal{U}), P\,X : \mathcal{U}$
- $\mathsf{pack} : \prod(X : \mathcal{U}), P\,X \to \exists(X : \mathcal{U}), P\,X$

Existential Types

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

- $\exists(X : \mathcal{U}), P\,X : \mathcal{U}$
- pack : $\prod(X : \mathcal{U}), P\,X \to \exists(X : \mathcal{U}), P\,X$

together with a recursion principle:

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

satisfying the expected β- and η-rules.

Encoding Streams

Let E be a type. We define Stream^* as follows[5].

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

This allows us to define:

- $\mathsf{hd}^* : \mathsf{Stream}^* \to E$
- $\mathsf{tl}^* : \mathsf{Stream}^* \to \mathsf{Stream}^*$
- $\mathsf{corec}^* : \prod (X : \mathcal{U}), (X \to E) \to (X \to X) \to X \to \mathsf{Stream}^*$

[5] Geuvers. "The Church-Scott representation of inductive and coinductive data"

Fixing the Impredicative Encoding for Streams

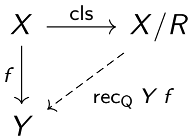
- Just like for lists, we cannot prove a suitable coinduction principle for Stream*.
- Fix for lists: take a subtype
- Fix for streams: take a **quotient**

Quotient Types

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

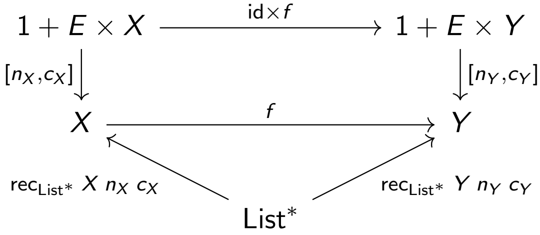
- a type $X/R : \mathcal{U}$
- a function $\mathsf{cls} : X \to X/R$

For all $Y : \mathcal{U}$ and $f : X \to Y$ that respects R, there is a unique $\mathsf{rec}_{\mathsf{Q}}\ Y\ f$ making the following diagram commute



Recall: Fixing Impredicative Encodings for Lists

To define $\mathrm{Lim}_{\mathrm{List}}$:
Suppose we have a commuting square.



Then the bottom triangle must commute.

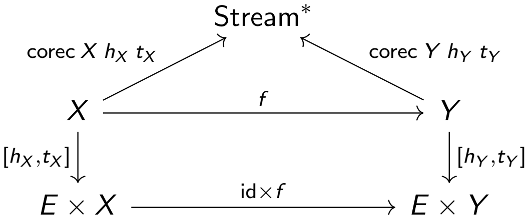
Recall: Fixing Impredicative Encodings for Streams

Suppose we have a commuting square.

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

Fixing Impredicative Encodings for Streams

Suppose we have a commuting square.



Then the upper triangle must commute

Fixing Impredicative Encodings for Streams

Given σ, τ : Stream*, we say $\sigma \equiv \tau$ if

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

Fixing Impredicative Encodings for Streams

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

$$\begin{aligned}
\exists &(X : \mathcal{U})(h_X : X \to E)(t_X : X \to X) \\
&(Y : \mathcal{U})(h_Y : Y \to E)(t_Y : Y \to Y) \\
&(f : X \to Y)
\end{aligned}$$

Fixing Impredicative Encodings for Streams

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

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

Fixing Impredicative Encodings for Streams

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

$$\begin{aligned}
\exists & (X : \mathcal{U})(h_X : X \to E)(t_X : X \to X) \\
& (Y : \mathcal{U})(h_Y : Y \to E)(t_Y : Y \to Y) \\
& (f : X \to Y) \\
& (p_h : \prod (x : X), h_X\ x = h_Y (f\ y)) \\
& (p_t : \prod (x : X), t_Y\ (f\ x) = f\ (t_X\ x)) \\
& (x : X), \\
& \sigma = \text{corec}\ X\ h_X\ t_X\ x \\
& \ \wedge \tau = \text{corec}\ Y\ h_Y\ t_Y\ (f\ x)
\end{aligned}$$

Fixing Impredicative Encodings for Streams

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

$$
\begin{aligned}
\exists & (X : \mathcal{U})(h_X : X \to E)(t_X : X \to X) \\
& (Y : \mathcal{U})(h_Y : Y \to E)(t_Y : Y \to Y) \\
& (f : X \to Y) \\
& (p_h : \prod(x : X), h_X\, x = h_Y (f\, y)) \\
& (p_t : \prod(x : X), t_Y\, (f\, x) = f\, (t_X\, x)) \\
& (x : X), \\
& \sigma = \text{corec}\, X\, h_X\, t_X\, x \\
& \wedge \tau = \text{corec}\, Y\, h_Y\, t_Y\, (f\, x)
\end{aligned}
$$

Define $\text{Stream} = \text{Stream}^* / \equiv$.

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 talk: demonstrate it for streams
- This method works for M-types

See our paper "Impredicative Encodings of Inductive and Coinductive Types" at FSCD2025

Existential Types

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

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

We define $\mathsf{Lim}_\exists$ similarly to $\mathsf{Lim}_{\mathsf{List}}$ and

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

Quotient Types

The starting point is the following type:

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

Here resp f R says that f respects R.

Quotient Types

The starting point is the following type:

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

Here resp f R says that f respects R.
We define Lim_Q similarly to $\mathsf{Lim}_\mathsf{List}$ and

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

Encoding Streams: Tails

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

$$\mathsf{tl}^*\ s = ?$$

where $? : \mathsf{Stream}^*$

Encoding Streams: Tails

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

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

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

Encoding Streams: Tails

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

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

where $? : \mathsf{Stream}^*$
Here:

- $Z : \mathcal{U}$
- $z : Z$
- $h : Z \to E$
- $t : Z \to Z$

Encoding Streams: Tails

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

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

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

- $Z : \mathcal{U}$
- $z : Z$
- $h : Z \to E$
- $t : Z \to Z$

Encoding Streams: Tails

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

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

Here:

- $Z : \mathcal{U}$
- $z : Z$
- $h : Z \to E$
- $t : Z \to Z$

Encoding Streams: Tails

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

$$\text{tl}^*\, s = \text{rec}_\exists\, \text{Stream}^*\, (\lambda Z\, z\, h\, t, \text{pack}\; Z\; \langle t\; z, h, t \rangle)\; s$$

Here:

- $Z : \mathcal{U}$
- $z : Z$
- $h : Z \to E$
- $t : Z \to Z$