Coinduction demystified

Valentin Robert

Galois, Inc

Assumptions about the audience

This talk assumes a bit of familiarity with *data type declarations*.

Assumptions about the audience

This talk assumes a bit of familiarity with *data type declarations*.

Familiarity with `GADT`s will help, but is not mandatory!

Assumptions about the audience

This talk assumes a bit of familiarity with *data type declarations*.

Familiarity with `GADT`s will help, but is not mandatory!

There will also be *inference rules*.

Why this talk?

When I first tried to learn about coinductive types,

I found many sentences I did not comprehend:

Why this talk?

When I first tried to learn about coinductive types,

I found many sentences I did not comprehend:

- “like inductive types, but the greatest fixed point”

Why this talk?

When I first tried to learn about coinductive types,

I found many sentences I did not comprehend:

- “like inductive types, but the greatest fixed point”
- “like inductive types, but read the rules backward”

Why this talk?

When I first tried to learn about coinductive types,

I found many sentences I did not comprehend:

- “like inductive types, but the greatest fixed point”
- “like inductive types, but read the rules backward”
- and the classic: “just the dual!”

Why this talk?

When I first tried to learn about coinductive types,

I found many sentences I did not comprehend:

- “like inductive types, but the greatest fixed point”
- “like inductive types, but read the rules backward”
- and the classic: “just the dual!”

Additionally, I was utterly confused by coinduction proof principles.

Why this talk?

When I first tried to learn about coinductive types,

I found many sentences I did not comprehend:

- “like inductive types, but the greatest fixed point”
- “like inductive types, but read the rules backward”
- and the classic: “just the dual!”

Additionally, I was utterly confused by coinduction proof principles.

I hope I can make it feel less inscrutable!

Let's talk about *induction*

In discrete mathematics, you may have learned about natural number induction, while trying to prove:

In discrete mathematics, you may have learned about natural number induction, while trying to prove:

$$\forall\ (n \in \mathtt{Nat}),\ P\ n$$

In discrete mathematics, you may have learned about natural number induction, while trying to prove:

```
∀ (n ∈ Nat), P n
```

and with it, a proof method along the lines of:

In discrete mathematics, you may have learned about natural number induction, while trying to prove:

$$\forall\ (n \in \mathtt{Nat}),\ P\ n$$

and with it, a proof method along the lines of:

- Prove $P\ \mathtt{0}$

In discrete mathematics, you may have learned about natural number induction, while trying to prove:

$$\forall\ (n \in \mathrm{Nat}),\ P\ n$$

and with it, a proof method along the lines of:

- Prove $P\ 0$
- Prove that for any number n,

In discrete mathematics, you may have learned about natural number induction, while trying to prove:

$$\forall\ (n \in \mathsf{Nat}),\ P\ n$$

and with it, a proof method along the lines of:

- Prove $P\ \mathsf{0}$
- Prove that for any number n,

  $P\ n$ implies $P\ (\mathsf{S}\ n)$

Then, you may have learned about lists in a programming class, and tried to prove some property:

Then, you may have learned about lists in a programming class, and tried to prove some property:

$$\forall\ (l \in \mathsf{List}\ T),\ P\ l$$

Then, you may have learned about lists in a programming class, and tried to prove some property:

$$\forall\ (l \in \mathtt{List}\ T),\ P\ l$$

and learned to prove it by induction:

Then, you may have learned about lists in a programming class, and tried to prove some property:

$$\forall\ (l \in \mathtt{List}\ T),\ P\ l$$

and learned to prove it by induction:

- Prove P `Nil`

Then, you may have learned about lists in a programming class, and tried to prove some property:

$$\forall\ (l \in \texttt{List}\ T),\ P\ l$$

and learned to prove it by induction:

- Prove P `Nil`
- Prove that for any head h, and for any tail t,

Then, you may have learned about lists in a programming class, and tried to prove some property:

$$\forall\ (l \in \mathtt{List}\ T),\ P\ l$$

and learned to prove it by induction:

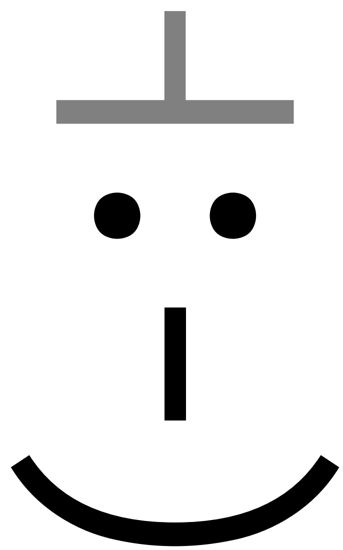
- Prove P $\mathtt{Nil}$
- Prove that for any head h, and for any tail t,

  $P\ t$ **implies** $P\ (\mathtt{Cons}\ h\ t)$

Why is this sane?

Let's put on our proof theory hats!

Let's put on our proof theory hats!



Inductive List T :=

$$\frac{}{\cdot \vdash x \in \texttt{List}\ T}$$

```
Inductive List (T : Type) :=
```

$$\frac{\cdot \vdash T \in \mathsf{Type}}{\cdot \vdash x \in \mathsf{List}\ T}$$

```
Inductive List (T : Type) ≔
| Nil  : List T
```

$$\frac{\cdot \vdash T \in \mathsf{Type} \qquad x \equiv \mathsf{Nil}}{\cdot \vdash x \in \mathsf{List}\ T}$$

```
Inductive List (T : Type) :=
| Nil  : List T
| Cons : (h : T) → (t : List T) → List T
.
```

$$\frac{\cdot \vdash T \in \mathsf{Type} \qquad x \equiv \mathsf{Nil}}{\cdot \vdash x \in \mathsf{List}\ T}$$

$$\frac{\cdot \vdash T \in \mathsf{Type} \qquad \cdot \vdash h \in T \qquad \cdot \vdash t \in \mathsf{List}\ T \qquad x \equiv \mathsf{Cons}\ h\ t}{\cdot \vdash x \in \mathsf{List}\ T}$$

This gives us three perspective on *inductive* types.

This gives us three perspective on *inductive* types.

Given a set of rules, to be read forward,

This gives us three perspective on *inductive* types.

Given a set of rules, to be read forward,

their values can be thought of as either:

This gives us three perspective on *inductive* types.

Given a set of rules, to be read forward,

their values can be thought of as either:

1. all values that have *finite* proofs using those rules

This gives us three perspective on *inductive* types.

Given a set of rules, to be read forward,

their values can be thought of as either:

1. all values that have *finite* proofs using those rules
2. the *smallest* set closed under those rules

This gives us three perspective on *inductive* types.

Given a set of rules, to be read forward,

their values can be thought of as either:

1. all values that have *finite* proofs using those rules
2. the *smallest* set closed under those rules
3. the *least* fixed point of the underlying endofunctor

This gives us three perspective on *inductive* types.

Given a set of rules, to be read forward,

their values can be thought of as either:

1. all values that have *finite* proofs using those rules
2. the *smallest* set closed under those rules
3. the *least* fixed point of the underlying endofunctor

I will give you an intuition for all three definitions now.

1. all values that have *finite* proofs using those rules

$$\frac{}{\cdot \vdash \texttt{Cons 0 Nil} \in \texttt{List Nat}}$$

1. all values that have *finite* proofs using those rules

$$\dfrac{\dfrac{}{\cdot \vdash \texttt{Nat} \in \texttt{Type}}}{\cdot \vdash \texttt{Cons 0 Nil} \in \texttt{List Nat}}$$

1. all values that have *finite* proofs using those rules

$$\dfrac{\dfrac{}{\cdot \vdash \texttt{Nat} \in \texttt{Type}} \qquad \dfrac{}{\cdot \vdash \texttt{0} \in \texttt{Nat}}}{\cdot \vdash \texttt{Cons 0 Nil} \in \texttt{List Nat}}$$

1. all values that have *finite* proofs using those rules

$$\dfrac{\dfrac{}{\cdot \vdash \texttt{Nat} \in \texttt{Type}} \qquad \dfrac{\dfrac{}{\cdot \vdash \texttt{Nat} \in \texttt{Type}}}{\cdot \vdash \texttt{0} \in \texttt{Nat}}}{\cdot \vdash \texttt{Cons 0 Nil} \in \texttt{List Nat}}$$

1. all values that have *finite* proofs using those rules

$$\dfrac{\dfrac{}{\cdot \vdash \texttt{Nat} \in \texttt{Type}} \qquad \dfrac{\dfrac{}{\cdot \vdash \texttt{Nat} \in \texttt{Type}}}{\cdot \vdash \texttt{0} \in \texttt{Nat}} \qquad \dfrac{}{\cdot \vdash \texttt{Nil} \in \texttt{List Nat}}}{\cdot \vdash \texttt{Cons 0 Nil} \in \texttt{List Nat}}$$

1. all values that have *finite* proofs using those rules

$$\dfrac{\dfrac{}{\cdot \vdash \texttt{Nat} \in \texttt{Type}} \qquad \dfrac{\dfrac{}{\cdot \vdash \texttt{Nat} \in \texttt{Type}}}{\cdot \vdash \texttt{0} \in \texttt{Nat}} \qquad \dfrac{\dfrac{}{\cdot \vdash \texttt{Nat} \in \texttt{Type}}}{\cdot \vdash \texttt{Nil} \in \texttt{List Nat}}}{\cdot \vdash \texttt{Cons 0 Nil} \in \texttt{List Nat}}$$

2. the *smallest* set closed under those rules

Let's try to compute `List Nat` starting with `{}`,
assuming `· ⊢ Nat ∈ Type` and `Nat = {0, 1, 2, ...}`

2. the *smallest* set closed under those rules

$$\texttt{List Nat} = \{\}$$

$$\frac{\cdot \vdash T \in \texttt{Type} \qquad x \equiv \texttt{Nil}}{\cdot \vdash x \in \texttt{List}\ T}$$

2. the *smallest* set closed under those rules

$$\mathtt{List\ Nat} = \{\}$$

$$\frac{\cdot \vdash \mathtt{Nat} \in \mathtt{Type} \qquad x \equiv \mathtt{Nil}}{\cdot \vdash x \in \mathtt{List\ Nat}}$$

2. the *smallest* set closed under those rules

$$\texttt{List Nat} = \{\}$$

$$\frac{\cdot \vdash \texttt{Nat} \in \texttt{Type} \qquad \texttt{Nil} \equiv \texttt{Nil}}{\cdot \vdash \texttt{Nil} \in \texttt{List Nat}}$$

2. the *smallest* set closed under those rules

$$\mathtt{List\ Nat} = \{\mathtt{Nil}\}$$

$$\frac{\cdot \vdash \mathtt{Nat} \in \mathtt{Type} \qquad \mathtt{Nil} \equiv \mathtt{Nil}}{\cdot \vdash \mathtt{Nil} \in \mathtt{List\ Nat}}$$

2. the *smallest* set closed under those rules

$$\mathtt{List\ Nat} = \{\mathtt{Nil}\}$$

$$\frac{\cdot \vdash T \in \mathtt{Type} \qquad \cdot \vdash h \in T \qquad \cdot \vdash t \in \mathtt{List}\ T}{\cdot \vdash \mathtt{Cons}\ h\ t \in \mathtt{List}\ T}$$

2. the *smallest* set closed under those rules

$$\texttt{List Nat} = \{\texttt{Nil}\}$$

$$\frac{\cdot \vdash \texttt{Nat} \in \texttt{Type} \qquad \cdot \vdash h \in \texttt{Nat} \qquad \cdot \vdash t \in \texttt{List Nat}}{\cdot \vdash \texttt{Cons}\ h\ t \in \texttt{List Nat}}$$

2. the *smallest* set closed under those rules

$$\texttt{List Nat} = \{\texttt{Nil}\}$$

$$\frac{\cdot \vdash \texttt{Nat} \in \texttt{Type} \qquad \cdot \vdash \texttt{0} \in \texttt{Nat} \qquad \cdot \vdash t \in \texttt{List Nat}}{\cdot \vdash \texttt{Cons 0}\ t \in \texttt{List Nat}}$$

2. the *smallest* set closed under those rules

$$\texttt{List Nat} = \{\texttt{Nil}\}$$

$$\frac{\cdot \vdash \texttt{Nat} \in \texttt{Type} \qquad \cdot \vdash \texttt{0} \in \texttt{Nat} \qquad \cdot \vdash \texttt{Nil} \in \texttt{List Nat}}{\cdot \vdash \texttt{Cons 0 Nil} \in \texttt{List Nat}}$$

2. the *smallest* set closed under those rules

$$\texttt{List Nat} = \{\texttt{Nil}, \texttt{Cons 0 Nil}\}$$

$$\frac{\cdot \vdash \texttt{Nat} \in \texttt{Type} \qquad \cdot \vdash \texttt{0} \in \texttt{Nat} \qquad \cdot \vdash \texttt{Nil} \in \texttt{List Nat}}{\cdot \vdash \texttt{Cons 0 Nil} \in \texttt{List Nat}}$$

2. the *smallest* set closed under those rules

```
List Nat = { Nil
```

2. the *smallest* set closed under those rules

```
List Nat  = { Nil
            , Cons 0 Nil
```

2. the *smallest* set closed under those rules

```
List Nat = { Nil
           , Cons 0 Nil
           , Cons 1 Nil
```

2. the *smallest* set closed under those rules

```
List Nat = { Nil
           , Cons 0 Nil
           , Cons 1 Nil
           , ...
```

2. the *smallest* set closed under those rules

```
List Nat = { Nil
           , Cons 0 Nil
           , Cons 1 Nil
           , ...
           , Cons 0 (Cons 0 Nil)
```

2. the *smallest* set closed under those rules

```
List Nat = { Nil
           , Cons 0 Nil
           , Cons 1 Nil
           , ...
           , Cons 0 (Cons 0 Nil)
           , Cons 1 (Cons 0 Nil)
```

2. the *smallest* set closed under those rules

```
List Nat = { Nil
           , Cons 0 Nil
           , Cons 1 Nil
           , ...
           , Cons 0 (Cons 0 Nil)
           , Cons 1 (Cons 0 Nil)
           , ...
           }
```

2. the *smallest* set closed under those rules

Now that's a pretty big *smallest* set...

2. the *smallest* set closed under those rules

Now that's a pretty big *smallest* set...

We will shortly see in what sense it could be larger!

3. the *least* fixed point of the underlying endofunctor

$F\ S =$

3. the *least* fixed point of the underlying endofunctor

$$F\ S = S \cup \{x \mid x \equiv \texttt{Nil}\}$$

3. the *least* fixed point of the underlying endofunctor

$$
\begin{aligned}
F\ S = \ & S \cup \{x \mid x \equiv \texttt{Nil}\} \\
& \cup \{x \mid h \in T
\end{aligned}
$$

3. the *least* fixed point of the underlying endofunctor

$$
\begin{aligned}
F\ S = S &\cup \{x \mid x \equiv \texttt{Nil}\} \\
&\cup \{x \mid h \in T \\
&\qquad\quad ,\ t \in \texttt{List}\ T
\end{aligned}
$$

3. the *least* fixed point of the underlying endofunctor

$$
\begin{aligned}
F\ S = S &\cup \{x \mid x \equiv \mathtt{Nil}\} \\
&\cup \{x \mid h \in T \\
&\qquad\quad ,\ t \in \mathtt{List}\ T \\
&\qquad\quad ,\ x \equiv \mathtt{Cons}\ h\ t \\
&\qquad\quad \}
\end{aligned}
$$

The *induction* proof method can therefore be abstracted as such:

The *induction* proof method can therefore be abstracted as such:

In order to prove $\forall\ x\ ,\ (x \in T) \Rightarrow (x \in P)$

The *induction* proof method can therefore be abstracted as such:

In order to prove $\forall\ x\ ,\ (x \in T) \Rightarrow (x \in P)$
by *induction* on T,

The *induction* proof method can therefore be abstracted as such:

In order to prove $\forall\ x\ ,\ (x \in T) \Rightarrow (x \in P)$
by *induction* on T,
it suffices to prove that P is closed under T's rules!

$$\frac{\cdot \vdash \mathsf{Nat} \in \mathsf{Type} \qquad x \equiv \mathsf{Nil}}{\cdot \vdash x \in \mathsf{List\ Nat}}$$

$$\frac{\cdot \vdash \mathsf{Nat} \in \mathsf{Type} \qquad \cdot \vdash h \in \mathsf{Nat} \qquad \cdot \vdash t \in \mathsf{List\ Nat} \qquad x \equiv \mathsf{Cons}\ h\ t}{\cdot \vdash x \in \mathsf{List\ Nat}}$$

$$\frac{\cdot \vdash \mathtt{Nat} \in \mathtt{Type} \qquad x \equiv \mathtt{Nil}}{\cdot \vdash x \in P}$$

$$\frac{\cdot \vdash \mathtt{Nat} \in \mathtt{Type} \qquad \cdot \vdash h \in \mathtt{Nat} \qquad \cdot \vdash t \in P \qquad x \equiv \mathtt{Cons}\ h\ t}{\cdot \vdash x \in P}$$

Proof-theoretic argument

The structure of the term guides the structure of a proof!

Proof-theoretic argument

The structure of the term guides the structure of a proof!

$$\dfrac{\dfrac{}{\cdot \vdash \mathtt{Nat} \in \mathtt{Type}} \qquad \dfrac{\dfrac{}{\cdot \vdash \mathtt{Nat} \in \mathtt{Type}}}{\cdot \vdash \mathtt{0} \in \mathtt{Nat}} \qquad \dfrac{\dfrac{}{\cdot \vdash \mathtt{Nat} \in \mathtt{Type}}}{\cdot \vdash \mathtt{Nil} \in \mathtt{List\ Nat}}}{\cdot \vdash \mathtt{Cons\ 0\ Nil} \in \mathtt{List\ Nat}}$$

Proof-theoretic argument

The structure of the term guides the structure of a proof!

$$\dfrac{\dfrac{}{\cdot \vdash \mathtt{Nat} \in \mathtt{Type}} \qquad \dfrac{\dfrac{}{\cdot \vdash \mathtt{Nat} \in \mathtt{Type}}}{\cdot \vdash \mathtt{0} \in \mathtt{Nat}} \qquad \dfrac{\dfrac{}{\cdot \vdash \mathtt{Nat} \in \mathtt{Type}}}{\cdot \vdash \mathtt{Nil} \in \mathtt{List\ Nat}}}{\cdot \vdash \mathtt{Cons\ 0\ Nil} \in \mathtt{List\ Nat}}$$

$$\dfrac{\dfrac{}{\cdot \vdash \mathtt{Nat} \in \mathtt{Type}} \qquad \dfrac{\dfrac{}{\cdot \vdash \mathtt{Nat} \in \mathtt{Type}}}{\cdot \vdash \mathtt{0} \in \mathtt{Nat}} \qquad \dfrac{\dfrac{}{\cdot \vdash \mathtt{Nat} \in \mathtt{Type}}}{\cdot \vdash \mathtt{Nil} \in P}}{\cdot \vdash \mathtt{Cons\ 0\ Nil} \in P}$$

Set-theoretic argument

Because T is the *smallest* set closed under its rules, $T \subseteq P$ and therefore $T = P$

Let's now consider *coinduction*

"Coinductive types model infinite structures unfolded on demand, like politicians' excuses: for each attack, there is a defence but no likelihood of resolution."

- Conor McBride

Let's think about *coinductive* types in the dual way.

Let's think about *coinductive* types in the dual way.

Given a set of rules, to be read backward,

Let's think about *coinductive* types in the dual way.

Given a set of rules, to be read backward,

their values can be thought of as either:

Let's think about *coinductive* types in the dual way.

Given a set of rules, to be read backward,

their values can be thought of as either:

1. all values that have *finite or infinite* proofs using those rules

Let's think about *coinductive* types in the dual way.

Given a set of rules, to be read backward,

their values can be thought of as either:

1. all values that have *finite or infinite* proofs using those rules

2. the *largest* set closed under those rules

Let's think about *coinductive* types in the dual way.

Given a set of rules, to be read backward,

their values can be thought of as either:

1. all values that have *finite or infinite* proofs using those rules
2. the *largest* set closed under those rules
3. the *greatest* fixed point of the underlying endofunctor

Read backward!?

$$\frac{x \equiv \mathtt{CoZero}}{\cdot \vdash x \in \mathtt{CoNat}}$$

$$\frac{\cdot \vdash n \in \mathtt{CoNat} \qquad x \equiv \mathtt{CoSucc}\ n}{\cdot \vdash x \in \mathtt{CoNat}}$$

Coinductive lists

$$\cfrac{\cdot \vdash T \in \mathtt{Type} \qquad x \equiv \mathtt{CoNil}}{\cdot \vdash x \in \mathtt{CoList}\ T}$$

$$\cfrac{\cdot \vdash T \in \mathtt{Type} \qquad \cdot \vdash h \in T \qquad \cdot \vdash t \in \mathtt{CoList}\ T \qquad x \equiv \mathtt{CoCons}\ h\ t}{\cdot \vdash x \in \mathtt{CoList}\ T}$$

Coinductive streams

$$\frac{\cdot \vdash T \in \mathtt{Type} \qquad \cdot \vdash h \in T \qquad \cdot \vdash t \in \mathtt{CoList}\ T \qquad x \equiv \mathtt{CoCons}\ h\ t}{\cdot \vdash x \in \mathtt{CoList}\ T}$$

The *coinduction* proof method can therefore be abstracted as such:

The *coinduction* proof method can therefore be abstracted as such:

In order to prove $\forall\ x\ ,\ (x \in P) \Rightarrow (x \in T)$

The *coinduction* proof method can therefore be abstracted as such:

In order to prove $\forall\ x\ ,\ (x \in P) \Rightarrow (x \in T)$
by *coinduction* on T,

The *coinduction* proof method can therefore be abstracted as such:

In order to prove $\forall\ x\ ,\ (x \in P) \Rightarrow (x \in T)$
by *coinduction* on T,
it suffices to prove that P is closed under T's rules!

The *coindaction* proof method can therefore be abstracted as such:

In order to prove $\forall\ x\ ,\ (x \in P) \Rightarrow (x \in T)$
by *coinduction* on T,
it suffices to prove that P is closed under T's rules!
(again, read *backward*)

Strengthening

When doing a proof by *induction*,
you might have had to strengthen your *inductive* hypothesis.

Strengthening

When doing a proof by *induction*,
you might have had to strengthen your *inductive* hypothesis.

$$\forall\ x\ ,\ (x \in T) \Rightarrow (x \in P)$$

Strengthening

When doing a proof by *induction*,
you might have had to strengthen your *inductive* hypothesis.

$$\forall\ x\ ,\ (x \in T) \Rightarrow (x \in P)$$

P was not closed under the *constructors* of T .

Strengthening

When doing a proof by *induction*,
you might have had to strengthen your *inductive* hypothesis.

$$\forall\ x\ ,\ (x \in T) \Rightarrow (x \in P)$$

P was not closed under the *constructors* of T .

Intuitively, your property was undershooting the type.

Strengthening

When doing a proof by *coinduction*,
you might have had to strengthen your *coinductive* hypothesis.

Strengthening

When doing a proof by *coinduction*,
you might have had to strengthen your *coinductive* hypothesis.

$$\forall\ x\ ,\ (x \in P) \Rightarrow (x \in T)$$

Strengthening

When doing a proof by *coinduction*,
you might have had to strengthen your *coinductive* hypothesis.

$$\forall\ x\ ,\ (x \in P) \Rightarrow (x \in T)$$

P was not closed under the *destructors* of T .

Strengthening

When doing a proof by ***coinduction***,
you might have had to strengthen your ***coinductive*** hypothesis.

$$\forall\ x\ ,\ (x \in P) \Rightarrow (x \in T)$$

P was not closed under the ***destructors*** of T .

Intuitively, your property was overshooting the type.

The lack of power of equality

$\equiv$ stands for "definitional equality".

It is the smallest congruence that typically includes:

$\equiv$ stands for "definitional equality".

It is the smallest congruence that typically includes:

- α: safe renaming of bound variables

$\equiv$ stands for "definitional equality".

It is the smallest congruence that typically includes:

- α: safe renaming of bound variables
- β: reduction of applied λ-terms

$\equiv$ stands for "definitional equality".

It is the smallest congruence that typically includes:

- α: safe renaming of bound variables
- β: reduction of applied λ-terms
- δ: unfolding of definitions

$\equiv$ stands for "definitional equality".

It is the smallest congruence that typically includes:

- α: safe renaming of bound variables
- β: reduction of applied λ-terms
- δ: unfolding of definitions
- ζ: reduction of let-bindings

≡ stands for "definitional equality".

It is the smallest congruence that typically includes:

- α: safe renaming of bound variables
- β: reduction of applied λ-terms
- δ: unfolding of definitions
- ζ: reduction of let-bindings
- η: **f ≡ λ x . f x**

≡ stands for "definitional equality".

It is the smallest congruence that typically includes:

- α: safe renaming of bound variables
- β: reduction of applied λ-terms
- δ: unfolding of definitions
- ζ: reduction of let-bindings
- η: $f \equiv \lambda\, x\, .\, f\, x$
- ι: reduction of:

≡ stands for "definitional equality".

It is the smallest congruence that typically includes:

- α: safe renaming of bound variables
- β: reduction of applied λ-terms
- δ: unfolding of definitions
- ζ: reduction of let-bindings
- η: f ≡ λ x . f x
- ι: reduction of:
    - pattern-matching over known constructs

$\equiv$ stands for "definitional equality".

It is the smallest congruence that typically includes:

- α: safe renaming of bound variables
- β: reduction of applied λ-terms
- δ: unfolding of definitions
- ζ: reduction of let-bindings
- η: $f \equiv \lambda\, x\, .\, f\, x$
- ι: reduction of:
    - pattern-matching over known constructs
    - fixpoints over known producers

≡ stands for "definitional equality".

It is the smallest congruence that typically includes:

- α: safe renaming of bound variables
- β: reduction of applied λ-terms
- δ: unfolding of definitions
- ζ: reduction of let-bindings
- η: f ≡ λ x . f x
- ι: reduction of:
  - pattern-matching over known constructs
  - fixpoints over known producers
  - cofixpoints under known consumers

Sadly, this does not capture everything we consider equal! For instance:

$$1 + n \not\equiv n + 1$$

Sadly, this does not capture everything we consider equal! For instance:

$$1 + n \not\equiv n + 1$$

Solution: propositional equality

Sadly, this does not capture everything we consider equal! For instance:

$$1 + n \not\equiv n + 1$$

Solution: propositional equality

$$\frac{T \in \mathsf{Type} \qquad x \in T}{x = x}$$

Sadly, this does not capture everything we consider equal! For instance:

$$1 + n \not\equiv n + 1$$

Solution: propositional equality

$$\frac{T \in \mathsf{Type} \qquad x \in T}{x = x}$$

Huh!?

Let us prove:

$$\forall\ (n \in \mathtt{Nat}),\ 1 + n = n + 1$$

by *induction* on `Nat`

Let us prove:

$$\forall\ (n \in \mathtt{Nat}),\ 1 + n = n + 1$$

by *induction* on `Nat`

$$\frac{}{1 + 0 = 0 + 1}$$

Let us prove:

$$\forall\ (n \in \mathtt{Nat}),\ 1 + n = n + 1$$

by *induction* on Nat

$$\dfrac{\dfrac{}{1 = 1}}{1 + 0 = 0 + 1}$$

Let us prove:

$$\forall\ (n \in \mathtt{Nat}),\ 1 + n = n + 1$$

by *induction* on Nat

$$\dfrac{\dfrac{\overline{1 \equiv 1}}{1 = 1}}{1 + 0 = 0 + 1}$$

Let us prove:

$$\forall\ (n \in \mathrm{Nat}),\ 1 + n = n + 1$$

by *induction* on Nat

$$\dfrac{\dfrac{\dfrac{}{1 \equiv 1}}{1 = 1}}{1 + 0 = 0 + 1}$$

$$\dfrac{}{1 + n = n + 1 \Rightarrow 1 + S\ n = S\ n + 1}$$

Let us prove:

$$\forall\ (n \in \mathrm{Nat}),\ 1 + n = n + 1$$

by *induction* on Nat

$$\dfrac{\dfrac{\dfrac{}{1 \equiv 1}}{1 = 1}}{1 + 0 = 0 + 1}$$

$$\dfrac{\dfrac{}{1 + n = n + 1 \Rightarrow S\ (S\ n) = S\ (n + 1)}}{1 + n = n + 1 \Rightarrow 1 + S\ n = S\ n + 1}$$

Let us prove:

$$\forall\ (n \in \mathtt{Nat}),\ 1 + n = n + 1$$

by *induction* on Nat

$$\dfrac{\dfrac{\dfrac{}{1 \equiv 1}}{1 = 1}}{1 + 0 = 0 + 1}$$

$$\dfrac{\dfrac{\dfrac{}{1 + n = n + 1 \Rightarrow S\ (S\ n) = S\ (1 + n)}}{1 + n = n + 1 \Rightarrow S\ (S\ n) = S\ (n + 1)}}{1 + n = n + 1 \Rightarrow 1 + S\ n = S\ n + 1}$$

Let us prove:

$$\forall\ (n \in \mathrm{Nat}),\ 1 + n = n + 1$$

by *induction* on Nat

$$\dfrac{\dfrac{\dfrac{}{1 \equiv 1}}{1 = 1}}{1 + 0 = 0 + 1}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{1 + n = n + 1 \Rightarrow S\ (S\ n) = S\ (S\ n)}}{1 + n = n + 1 \Rightarrow S\ (S\ n) = S\ (1 + n)}}{1 + n = n + 1 \Rightarrow S\ (S\ n) = S\ (n + 1)}}{1 + n = n + 1 \Rightarrow 1 + S\ n = S\ n + 1}$$

Let us prove:

$$\forall\ (n \in \mathrm{Nat}),\ 1 + n = n + 1$$

by ***induction*** **on** Nat

$$\dfrac{\dfrac{\overline{1 \equiv 1}}{1 = 1}}{1 + 0 = 0 + 1}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{1 + n = n + 1 \Rightarrow \mathrm{S}\ (\mathrm{S}\ n) \equiv \mathrm{S}\ (\mathrm{S}\ n)}}{1 + n = n + 1 \Rightarrow \mathrm{S}\ (\mathrm{S}\ n) = \mathrm{S}\ (\mathrm{S}\ n)}}{1 + n = n + 1 \Rightarrow \mathrm{S}\ (\mathrm{S}\ n) = \mathrm{S}\ (1 + n)}}{1 + n = n + 1 \Rightarrow \mathrm{S}\ (\mathrm{S}\ n) = \mathrm{S}\ (n + 1)}}{1 + n = n + 1 \Rightarrow 1 + \mathrm{S}\ n = \mathrm{S}\ n + 1}$$

More values, more problems!

Co-fixpoints are not judgmentally equal either! :-(

```
comap (+ 1) zeroes ≢ ones
comap (+ 1) zeroes ≠ ones
```

More values, more problems!

Co-fixpoints are not judgmentally equal either! :-(

$$\texttt{comap}\ (+\ 1)\ \mathit{zeroes} \not\equiv \mathit{ones}$$

$$\texttt{comap}\ (+\ 1)\ \mathit{zeroes} \neq \mathit{ones}$$

Solutions

More values, more problems!

Co-fixpoints are not judgmentally equal either! :-(

$$\mathtt{comap}\ (+\ 1)\ \mathit{zeroes} \not\equiv \mathit{ones}$$

$$\mathtt{comap}\ (+\ 1)\ \mathit{zeroes} \not\equiv \mathit{ones}$$

Solutions

- **Build an *inductive* argument for finite observations**

More values, more problems!

Co-fixpoints are not judgmentally equal either! :-(

```
comap (+ 1) zeroes ≢ ones
comap (+ 1) zeroes ≠ ones
```

Solutions

- Build an *inductive* argument for finite observations
- Build a *coinductive* argument for infinite observations

Stream "equality"

$$\frac{h_1 \equiv h_2 \qquad t_1 \approx t_2}{\mathsf{CoCons}\ h_1\ t_1 \approx \mathsf{CoCons}\ h_2\ t_2}$$

In practice, the coinduction principle I presented
earlier is rarely what we are interested in.
Many proofs on *coinductive* data types are instead relational.

In practice, the coinduction principle I presented earlier is rarely what we are interested in.
Many proofs on *coinductive* data types are instead relational.

Bisimulation techniques demonstrate how some binary relations are closed under the *destructors* of a type.

In practice, the coinduction principle I presented earlier is rarely what we are interested in.
Many proofs on *coinductive* data types are instead relational.

Bisimulation techniques demonstrate how some binary relations are closed under the *destructors* of a type.

There is a dual proof technique, *congruence*,
capturing binary relations closed under the *constructors* of a type.

Finally, because this is a Galois talk:

Finally, because this is a Galois talk:

There is yet another, categorical, interpretation of

inductive and *coinductive* types.

Inductive types are *initial* algebras:

$$\mathtt{T\ A} \longrightarrow \mathtt{A}$$

Inductive types are *initial* algebras:

$$\mathtt{T}\ \mathtt{A} \longrightarrow \mathtt{A}$$

of functors like:

Inductive types are *initial* algebras:

$$T\ A \longrightarrow A$$

of functors like:

$T\ X = 1 + X$ (natural numbers)

Inductive types are *initial* algebras:

$$\mathsf{T}\ \mathsf{A} \longrightarrow \mathsf{A}$$

of functors like:

$\mathsf{T}\ \mathsf{X} = 1 + \mathsf{X}$ (natural numbers)

$\mathsf{T}\ \mathsf{X} = 1 + \mathsf{T} \times \mathsf{X}$ (lists of Ts)

Coinductive types are *final co*algebras:

$$A \longrightarrow T\ A$$

Coinductive types are *final co*algebras:

$$A \longrightarrow T\ A$$

of functors like:

Coinductive types are *final co*algebras:

$$A \longrightarrow T\ A$$

of functors like:

$T\ X = 1 + X$ (predecessor function)

Coinductive types are *final co*algebras:

$$A \longrightarrow T\ A$$

of functors like:

$T\ X = 1 + X$ (predecessor function)

$T\ X = 1 + T \times X$ (possibly-finite streams of Ts)

Great things about the categorical intuition

$$\mathtt{T\ A} \longrightarrow \mathtt{A} \qquad\qquad \mathtt{A} \longrightarrow \mathtt{T\ A}$$

Great things about the categorical intuition

$$\mathtt{T\ A} \longrightarrow \mathtt{A} \qquad\qquad \mathtt{A} \longrightarrow \mathtt{T\ A}$$

- Gives a great intuition for the constructor / destructor roles

Great things about the categorical intuition

$$\mathtt{T\ A} \longrightarrow \mathtt{A} \qquad\qquad \mathtt{A} \longrightarrow \mathtt{T\ A}$$

- Gives a great intuition for the constructor / destructor roles
- The duality story is very crisp

Great things about the categorical intuition

$$\mathtt{T\ A} \longrightarrow \mathtt{A} \qquad\qquad \mathtt{A} \longrightarrow \mathtt{T\ A}$$

- Gives a great intuition for the constructor / destructor roles
- The duality story is very crisp
- Justifies case analysis by the existence of a homomorphism

Great things about the categorical intuition

$$\mathtt{T\ A} \longrightarrow \mathtt{A} \qquad\qquad \mathtt{A} \longrightarrow \mathtt{T\ A}$$

- Gives a great intuition for the constructor / destructor roles
- The duality story is very crisp
- Justifies case analysis by the existence of a homomorphism
- Justifies *induction* and *coinduction* by the uniqueness of said homomorphism

Learn more!

Jacobs, B. and Rutten, J., 1997.
A tutorial on (co) algebras and (co) induction.
Bulletin-European Association for Theoretical Computer Science, 62, pp.222-259.