

Implementation of Type Theory based on dependent Inductive and Coinductive Types

Florian Engel

November 20, 2020

Inductive types

- ▶ defined over their constructors
- ▶ each constructor has to give back this type

Examples

- ▶ Booleans

```
data Bool : Set where
  True  : Unit → Bool
  False : Unit → Bool
```

- ▶ Natural numbers

```
data Nat : Set where
  Zero : Unit → Nat
  Succ : Nat → Nat
```

Destructing inductive types

- ▶ also called recursion
- ▶ pattern matches on the constructor
- ▶ gives back values of same type in each match

Examples

- ▶ negation

```
rec Bool to Bool where
  True u = False @ ◇
  False u = True @ ◇
```

- ▶ isZero

```
rec Nat to Bool where
  Zero u = True @ ◇
  Succ n = False @ ◇
```

positive coinductive types

- ▶ treats recursive occurrence like a value
- ▶ otherwise like inductive types
- ▶ functions which produce such types have to be productive

Example

- ▶ Stream in coq

```
CoInductive Stream (A : Set) : Set :=  
  Cons : A -> Stream A -> Stream A.
```

- ▶ repeat function

```
CoFixpoint repeat (A : Set) (x : A) : Stream A :=  
  Cons A x (repeat A x).
```

What is wrong about positive coinductive types

- ▶ Symmetry between with inductive types not clear
- ▶ Breaks subject reduction

negative coinductive types

- ▶ defined over their destructors
- ▶ functions use copattern matching

Examples

- ▶ Stream

```
codata Stream⟨A : Set⟩ : Set where
  Hd : Stream → A
  Tl : Stream → Stream
```

- ▶ repeat function

```
repeat⟨A : Set⟩(x : A) =
  corec Unit to Stream⟨A⟩ where
    { Hd s = x
      ; Tl s = ◇ } @ ◇
```

Type Theory based on dependent Inductive and Coinductive Types

- ▶ kinds: $(x_1 : A_1, \dots, x_n : A_n) \rightarrow *$
- ▶ types: $(x_1 : A_1, \dots, x_n : A_n) \rightarrow B$
- ▶ lambda abstraction: $(x).A$
- ▶ type application: $A@t$
- ▶ term application: $t@s$
- ▶ inductive types: $\mu(X : \Gamma \rightarrow *; \vec{\sigma}; \vec{A})$
- ▶ coinductive types $\nu(X : \Gamma \rightarrow *; \vec{\sigma}; \vec{A})$
- ▶ recursion: $\text{rec } \overrightarrow{(\Gamma_k, y_k).g_k}$
- ▶ corecursion: $\text{corec } \overrightarrow{(\Gamma_k, y_k).g_k}$

Dependent coinductive types

Demo

Other topics in the thesis

- ▶ Termination and productivity checking with sized types
- ▶ Implementation details
 - ▶ Rules rewriting
 - ▶ De-Brujin indexes