

---

# M-types and Coinduction in HoTT and Cubical Type Theory

Lasse Letager Hansen, 201912345

---

Master's Thesis, Computer Science

March 9, 2020

Advisor: Bas Spitters



# Abstract

in English...



# Resumé

in Danish...



# Acknowledgments

...

*Lasse Letager Hansen,  
Aarhus, March 9, 2020.*





# Contents

<b>Abstract</b>	<b>iii</b>
<b>Resumé</b>	<b>v</b>
<b>Acknowledgments</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 M-types</b>	<b>3</b>
2.1 Containers / Signatures . . . . .	3
2.2 ITrees as M-types . . . . .	4
2.2.1 Delay Monad . . . . .	4
2.2.2 Tree . . . . .	5
2.2.3 ITrees . . . . .	6
2.3 Co-induction Principle for M-types . . . . .	6
2.3.1 Bisimulation of ITrees . . . . .	7
2.4 Examples of fixed points . . . . .	7
2.5 Quotient M-type . . . . .	7
2.6 Bisimulation . . . . .	8
2.7 Closure properties of M-types . . . . .	8
2.7.1 Closure under products . . . . .	8
<b>3 Additions to the Cubical Agda Library</b>	<b>9</b>
3.1 Lemma 10 . . . . .	10
<b>4 Conclusion</b>	<b>11</b>
<b>Bibliography</b>	<b>13</b>
<b>A The Technical Details</b>	<b>15</b>



# Chapter 1

## Introduction

motivate and explain the problem to be addressed

example of a citation: [1]

get your bibtex entries from <https://dblp.org/>



# Chapter 2

## M-types

### 2.1 Containers / Signatures

A Container (or Signature) is a pair  $\mathcal{S} = (\mathcal{A}, \mathcal{B})$  of types  $\vdash \mathcal{A} : \mathcal{U}$  and  $a : \mathcal{A} \vdash \mathcal{B}(a) : \mathcal{U}$ . From a container we can define a polynomial functor, defined for objects (types) as

$$P_{\mathcal{S}} : \mathcal{U} \rightarrow \mathcal{U}$$

$$P(\mathcal{X}) := P_{\mathcal{S}}(\mathcal{X}) = \sum_{a:\mathcal{A}} \mathcal{B}(a) \rightarrow \mathcal{X} \quad (2.1)$$

and for a function  $f : \mathcal{X} \rightarrow \mathcal{Y}$  as

$$Pf : P\mathcal{X} \rightarrow P\mathcal{Y}$$

$$Pf(a, g) = (a, f \circ g) \quad (2.2)$$

As an example lets look at type for streams over the type  $A$ , defined using the container  $\mathcal{S} = (\mathcal{A}, \mathbf{1})$ , applying the polynomial functor we get

$$P_{\mathcal{S}}(\mathcal{X}) = \sum_{a:A} \mathbf{1} \rightarrow \mathcal{X} \quad (2.3)$$

since we are working in a Category with exponentials we get  $\mathbf{1} \rightarrow \mathcal{X} \equiv \mathcal{X}^{\mathbf{1}} \equiv \mathcal{X}$ , furthermore  $\mathbf{1}$  and  $\mathcal{X}$  does not depend on  $A$  here, so this will be equivalent to the definition

$$P_{\mathcal{S}}(\mathcal{X}) = A \times \mathcal{X} \quad (2.4)$$

Now we define the coalgebra for this functor with type

$$\mathbf{Coalg}_{\mathcal{S}} = \sum_{C:\mathcal{U}} C \rightarrow P C \quad (2.5)$$

and morphisms

$$\_ \Rightarrow \_ : \mathbf{Coalg}_{\mathcal{S}} \rightarrow \mathbf{Coalg}_{\mathcal{S}}$$

$$(C, \gamma) \Rightarrow (D, \delta) = \sum_{f:C \rightarrow D} \delta \circ f = Pf \circ \gamma \quad (2.6)$$

M-types can now be defined from a container  $S$  as the type  $\mathbf{M}$  such that  $(\mathbf{M}, \text{out} : \mathbf{M} \rightarrow P_S \mathbf{M})$  fulfills the property

$$\text{Final}_S := \sum_{(X, \rho) : \text{Coalg}_S(C, \gamma)} \prod_{(C, \gamma) : \text{Coalg}_S} \text{isContr}((C, \gamma) \Rightarrow (X, \rho)) \quad (2.7)$$

that is  $\prod_{(C, \gamma) : \text{Coalg}_S} \text{isContr}((C, \gamma) \Rightarrow (\mathbf{M}, \text{out}))$ . We denote this construction of the type  $\mathbf{M}$ , as  $\mathbf{M}(A, B)$  or  $\mathbf{M}S$ .

If we continue our example for streams this will give us the M-type, we can see that  $P_S(\mathbf{M}) = A \times \mathbf{M}$ , meaning we have the following diagram, where  $\text{out}$  is an isomorphism (because of the finality of

$$\begin{array}{ccccc} A & \xleftarrow{\pi_1} & A \times \mathbf{M} & \xrightarrow{\pi_2} & \mathbf{M} \\ & \searrow \text{hd} & \uparrow \text{out} & \nearrow \text{tl} & \\ & & \mathbf{M} & & \end{array}$$

Figure 2.1: M-types of streams

the coalgebra), with inverse  $\text{in} : P_S \mathbf{M} \rightarrow \mathbf{M}$ . We now have a semantic for the rules we would expect for streams, if we let  $\text{cons} = \text{in}$  and  $\text{Stream } A = \mathbf{M}(A, \mathbf{1})$ ,

$$\frac{A : \mathcal{U} \quad s : \text{Stream } A}{\text{hd } s : A} \text{E}_{\text{hd}} \quad (2.8)$$

$$\frac{A : \mathcal{U} \quad s : \text{Stream } A}{\text{tl } s : \text{Stream } A} \text{E}_{\text{tl}} \quad (2.9)$$

$$\frac{A : \mathcal{U} \quad x : A \quad xs : \text{Stream } A}{\text{cons } x \ xs : \text{Stream } A} \text{I}_{\text{cons}} \quad (2.10)$$

## 2.2 ITrees as M-types

We want the following rules for ITrees

$$\frac{r : R}{\text{Ret } r : \text{itree } E R} \text{I}_{\text{Ret}} \quad (2.11)$$

$$\frac{A : \mathcal{U} \quad a : E A \quad f : A \rightarrow \text{itree } E R}{\text{Vis } a \ f : \text{itree } E R} \text{I}_{\text{Vis}}. \quad (2.12)$$

Elimination rules

$$\frac{t : \text{itree } E R}{\text{Tau } t : \text{itree } E R} \text{E}_{\text{Tau}}. \quad (2.13)$$

### 2.2.1 Delay Monad

We start by looking at **itrees** without the **Vis** constructor, this type is also know as the delay monad. We say this type is given by  $S = (\mathbf{1} + R, \lambda\{\text{inl } \_ \rightarrow \mathbf{1} ; \text{inr } \_ \rightarrow \mathbf{0}\})$  equal to  $\mathbf{M}S$ , we then get the polynomial functor

$$P_S(X) = \sum_{x : \mathbf{1} + R} \lambda\{\text{inl } \_ \rightarrow \mathbf{1} ; \text{inr } \_ \rightarrow \mathbf{0}\} x \rightarrow X \quad (2.14)$$

check  
this  
state-  
ment

This type is equal to the type:

$$P_S(X) = X + R \times (\mathbf{0} \rightarrow X) \quad (2.15)$$

we know that  $\mathbf{0} \rightarrow X \equiv \mathbf{1}$ , so we can further reduce to

$$P_S(X) = X + R \quad (2.16)$$

meaning we get the following diagram. What this diagram says is that we can define the operations

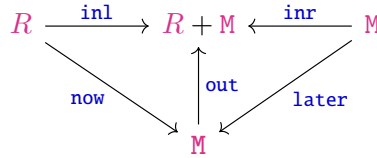


Figure 2.2: Delay monad

**now** and **later** using  $\mathbf{in} = \mathbf{out}^{-1}$  together with the injections **inl** and **inr**.

(Later  
= Tau,  
Ret =  
Now)

### 2.2.2 Tree

Now lets look at the example where we remove the **Tau** constructor. We let

$$S = \left( R + \sum_{A:\mathcal{U}} E A, \lambda\{\mathbf{inl} \_ \rightarrow \mathbf{0} ; \mathbf{inr} (A, e) \rightarrow A\} \right). \quad (2.17)$$

This will give us the polynomial functor:

$$P_S(X) = \sum_{x:R + \sum_{A:\mathcal{U}} E A} \lambda\{\mathbf{inl} \_ \rightarrow \mathbf{0} ; \mathbf{inr} (A, e) \rightarrow A\} x \rightarrow X \quad (2.18)$$

which simplifies to

$$P_S(X) = (R \times (\mathbf{0} \rightarrow X)) + (\sum_{A:\mathcal{U}} E A \times (A \rightarrow X)) \quad (2.19)$$

and further

$$P_S(X) = R + \sum_{A:\mathcal{U}} E A \times (A \rightarrow X) \quad (2.20)$$

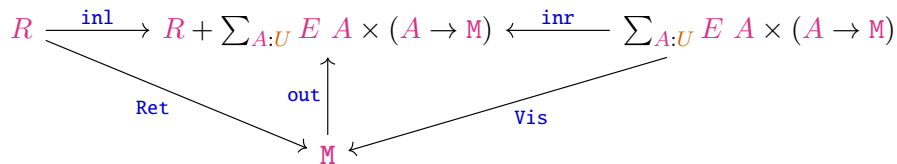


Figure 2.3: TODO: ???

Again we can define **Ret** and **Vis** using the **in** functor.

### 2.2.3 ITrees

Now we should have all the knowledge needed to make ITrees using  $\mathbf{M}$ -types. We define ITrees by the container:

$$S = \left( \mathbf{1} + R + \sum_{A:\mathcal{U}} (E\ A) \ , \ \lambda \{ \text{inl}(\text{inl}\ \_) \rightarrow \mathbf{1} ; \text{inl}(\text{inr}\ \_) \rightarrow \mathbf{0} ; \text{inr}(A, \_) \rightarrow A \} \right) \quad (2.21)$$

Then the (reduced) polynomial functor becomes

$$P_S(X) = X + R + \sum_{A:\mathcal{U}} ((E\ A) \times (A \rightarrow X)) \quad (2.22)$$

Giving us the diagram

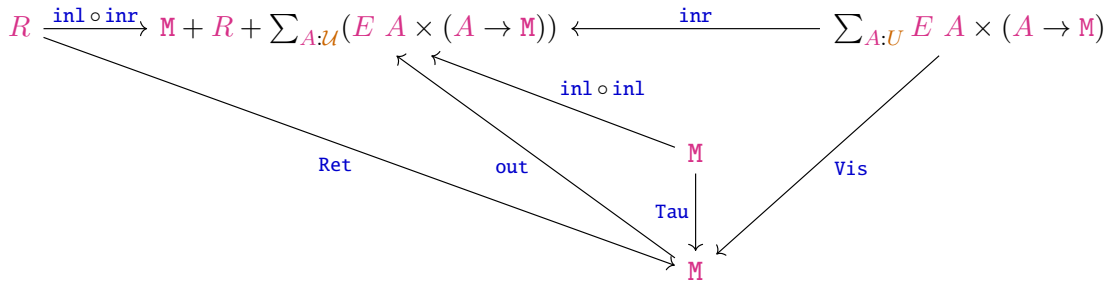


Figure 2.4: TODO: ???

## 2.3 Co-induction Principle for M-types

We can now construct a bisimulation: for all coalgebras  $C-\gamma : \text{Coalg}_S$ , if we have a relation  $\mathcal{R} : C \rightarrow C \rightarrow \mathcal{U}$ , and a type  $\overline{\mathcal{R}} = \sum_{a:C} \sum_{b:C} \mathcal{R}\ a\ b$ , such that  $\overline{\mathcal{R}}$  and  $\alpha_{\overline{\mathcal{R}}} : \overline{\mathcal{R}} \rightarrow P(\overline{\mathcal{R}})$  makes a  $P$ -coalgebra  $\overline{\mathcal{R}}-\alpha_{\overline{\mathcal{R}}} : \text{Coalg}_S$ , such that the following diagram commutes (where  $\Rightarrow$  are  $P$ -coalgebra morphisms).

$$C-\gamma \xleftarrow{\pi_1 \overline{\mathcal{R}}} \overline{\mathcal{R}}-\alpha_{\overline{\mathcal{R}}} \xrightarrow{\pi_2 \overline{\mathcal{R}}} C-\gamma$$

Furthermore for any bisimulation over a final  $P$ -coalgebra  $\mathbf{M}-\text{out} : \text{Coalg}_S$  we have the following diagram,

$$\mathbf{M}-\text{out} \xleftarrow{\pi_1 \overline{\mathcal{R}}} \overline{\mathcal{R}}-\alpha_{\overline{\mathcal{R}}} \xrightarrow{\pi_2 \overline{\mathcal{R}}} \mathbf{M}-\text{out}$$

where  $\pi_1 \overline{\mathcal{R}} = ! = \pi_2 \overline{\mathcal{R}}$ , which means given  $r : \mathcal{R}(m, m')$  we get  $m = \pi_1 \overline{\mathcal{R}}(m, m', r) = \pi_2 \overline{\mathcal{R}}(m, m', r) = m'$ .

We want to define a co-induction principle from any bisimulation relation over a final coalgebra, that is if  $R$  gives a bisimulation, then it is true that

$$R \equiv \equiv \quad (2.23)$$



meaning we can use the relation  $R$ , to show that two things of an  $\mathbf{M}$ -type are equivalent. So we want to construct an isomorphism between  $R$  and the equivalence relation  $\equiv$ , to do this we must construct functions

$$p : R \rightarrow \equiv \quad (2.24)$$

$$q : \equiv \rightarrow R \quad (2.25)$$

and relations

$$\alpha : p \circ q \equiv \text{id}_{\equiv} \quad (2.26)$$

$$\beta : q \circ p \equiv \text{id}_R \quad (2.27)$$

Complete the construction of equality from any bisimulation relation over an  $\mathbf{M}$ -type

### 2.3.1 Bisimulation of ITrees

We define our bisimulation coalgebra from the strong bisimulation relation  $\mathcal{R}$ , defined by the following rules.

$$\frac{a, b : R \quad a \equiv_R b}{\text{Ret } a \cong \text{Ret } b} \text{EqRet} \quad (2.28)$$

$$\frac{t, u : \text{itree } E \ R \quad t \cong u}{\text{Tau } t \cong \text{Tau } u} \text{EqTau} \quad (2.29)$$

$$\frac{A : \mathcal{U} \quad e : E \ A \quad k_1, k_2 : A \rightarrow \text{itree } E \ R \quad t \cong u}{\text{Vis } e \ k_1 \cong \text{Tau } e \ k_2} \text{EqVis} \quad (2.30)$$

Now we just need to define  $\alpha_{\overline{\mathcal{R}}}$ . Now we have a bisimulation relation, which is equivalent to equality, using what we showed in the previous section.

define  
the  $\alpha_{\overline{\mathcal{R}}}$   
function

## 2.4 Examples of fixed points

We want to define `spin`, as being the fixed point `spin = later spin`, so that is again a final coalgebra, but of a  $\mathbf{M}$ -type (which is a final coalgebra)

$$\begin{array}{c} \text{spin} \\ \downarrow \text{later} \\ \text{later spin} \end{array}$$

Since it is final, it also must be unique, meaning that there is just one program that spins forever, without returning a value, meaning every other program must return a value. If we just

## 2.5 Quotient M-type

Since we know that  $\mathbf{M}$ -types preserves the H-level, we can use set-truncated quotients, to define quotient  $\mathbf{M}$ -types, for examples we can define weak bisimulation of the delay monad as

## 2.6 Bisimulation

We want to define a bisimulation principle for M-types

## 2.7 Closure properties of M-types

### 2.7.1 Closure under products

The product of two M-types is again an M-type

$$\begin{array}{ccccccc}
 \mathbf{1} & \longrightarrow & P_{S_0} \mathbf{1} & \longrightarrow & \dots & \longrightarrow & P_{S_0}^n \mathbf{1} \longrightarrow \dots \\
 \uparrow & & \uparrow & & & & \uparrow \\
 \mathbf{1} \times \mathbf{1} & \longrightarrow & P_{S_0} \mathbf{1} \times P_{S_1} \mathbf{1} & \longrightarrow & \dots & \longrightarrow & P_{S_0}^n \mathbf{1} \times P_{S_1}^n \mathbf{1} \longrightarrow \dots \\
 \downarrow & & \downarrow & & & & \downarrow \\
 \mathbf{1} & \longrightarrow & P_{S_1} \mathbf{1} & \longrightarrow & \dots & \longrightarrow & P_{S_1}^n \mathbf{1} \longrightarrow \dots
 \end{array}$$

$$\mathbf{1} \longrightarrow P_{S_0 \times S_1} \mathbf{1} \longrightarrow \dots \longrightarrow P_{S_0 \times S_1}^n \mathbf{1} \longrightarrow \dots$$

$$\begin{array}{ccc}
 & A \times B & \\
 \nearrow^{hd \times hd} & & \nwarrow^{hd} \\
 Stream A \times Stream B & \xrightarrow{zip} & Stream(A \times B) \\
 \downarrow^{tl \times tl} & & \downarrow^{tl} \\
 Stream A \times Stream B & \xrightarrow{zip} & Stream(A \times B)
 \end{array}$$

From this we get the computation rules

$$hd \times hd \equiv hd \circ zip \quad (2.31)$$

$$zip \circ tl \times tl \equiv tl \circ zip \quad (2.32)$$



## Chapter 3

# Additions to the Cubical Agda Library

### 3.1 Lemma 10

$\mathbb{M}$ -types are part of a final coalgebra, formally  $\forall \textcolor{red}{S} \textcolor{red}{C} - \gamma, (\textcolor{red}{C} - \gamma \Rightarrow \mathbb{M} - \text{out}) \equiv \mathbf{1}$

$$U \equiv \sum_{f:C \rightarrow \mathcal{L}} \text{out} \circ f \equiv \text{step } f \quad (3.1)$$

$$\equiv \sum_{f:C \rightarrow \mathcal{L}} \text{in} \circ \text{out} \circ f \equiv \text{in} \circ \text{step } f \quad (3.2)$$

$$\equiv \sum_{f:C \rightarrow \mathcal{L}} f \equiv \text{in} \circ \text{step } f \quad (3.3)$$

$$\equiv \sum_{f:C \rightarrow \mathcal{L}} f \equiv \Psi f \quad (3.4)$$

$$\equiv \sum_{c:\text{Cone}} e \ c \equiv \Psi (e \ c) \quad (3.5)$$

$$\equiv \sum_{c:\text{Cone}} e \ c \equiv e \ (\phi \ c) \quad (3.6)$$

$$\equiv \sum_{c:\text{Cone}} c \equiv \phi \ c \quad (3.7)$$

$$\equiv \sum_{(u,q):\text{Cone}} (u, q) \equiv (\phi_0 \ u, \phi_1 \ u \ q) \quad (3.8)$$

$$\equiv \sum_{(u,q):\text{Cone}} \sum_{p:u \equiv \phi_0 \ u} q \equiv_{\lambda i, \text{Cone}_1(p \ i)} \phi_1 \ u \ q \quad (3.9)$$

$$\equiv \sum_{(u,p):\sum_{u:\text{Cone}_0} u \equiv \phi_0 \ u} \sum_{q:\text{Cone}_1 \ u} q \equiv_{\lambda i, \text{Cone}_1(p \ i)} \phi_1 \ u \ q \quad (3.10)$$

$$\equiv \sum_{q:\text{Cone}_1 u_0} q \equiv_{\lambda i, \text{Cone}_1(\text{funExt } p_0 \ i)} \phi_1 \ u_0 \ q \quad (3.11)$$

$$\vdots \quad (3.12)$$

$$\equiv \mathbf{1} \quad (3.13)$$

## Chapter 4

# Conclusion

conclude on the problem statement from the introduction



# Bibliography

- [1] Amin Timany and Matthieu Sozeau. Cumulative inductive types in coq. *LIPICs: Leibniz International Proceedings in Informatics*, 2018.





## Appendix A

# The Technical Details

