M-types and Coinduction in HoTT and Cubical Type Theory

Lasse Letager Hansen, 201912345

Master's Thesis, Computer Science

March 8, 2020

Advisor: Bas Spitters



Abstract

in English...

Resumé

in Danish...

${\bf Acknowledgments}$

Lasse Letager Hansen, Aarhus, March 8, 2020.

Contents

Abstract Resumé			ii
			\mathbf{v}
A	ckno	vledgments	ii
1	Inti	oduction	1
2	M-t	ypes Containers / Signatures	3
	2.2	ITrees as M-types	4
		v	4 5
	2.3		6
		2.3.1 Bisimulation of ITrees	7
	$\frac{2.4}{2.5}$	Quotient M-type	7
	2.6	Bisimulation	8
3		1.010125 00 0110 0 distout 11 9 dd 2151d1,	9
4 Conclusion		1	
Bibliography			.3
A	The	Technical Details	5



Introduction

motivate and explain the problem to be addressed

example of a citation: [1]

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

M-types

2.1 Containers / Signatures

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

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

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

and for a function $f: X \to Y$ as

$$Pf: PX \to PY$$

$$Pf(a,q) = (a, f \circ q)$$
(2.2)

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

$$P_{\mathbf{S}}(X) = \sum_{a:A} \mathbf{1} \to X \tag{2.3}$$

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

$$P_{\mathbf{S}}(X) = A \times X \tag{2.4}$$

Now we define the coalgebra for this functor with type

$$\mathsf{Coalg}_{S} = \sum_{C:\mathcal{U}} C \to PC \tag{2.5}$$

and morphisms

$$_\Rightarrow_: \mathtt{Coalg}_S \to \mathtt{Coalg}_S$$

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

$$(2.6)$$

M-types can now be defined from a container S as the type M such that $(M, out : M \to P_SM)$ fulfills the property

$$\mathtt{Final}_{\mathbf{S}} := \sum_{(X,\rho): \mathtt{Coalg}_{\mathbf{S}}} \prod_{(C,\gamma): \mathtt{Coalg}_{\mathbf{S}}} \mathtt{isContr}((C,\gamma) \Rightarrow (X,\rho)) \tag{2.7}$$

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

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

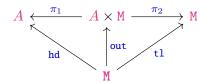


Figure 2.1: M-types of streams

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

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

$$\frac{A: \mathcal{U} \quad s: \mathtt{Stream} \ A}{\mathtt{tl} \ s: \mathtt{Stream} \ A} \ \mathtt{E}_{\mathtt{tl}} \tag{2.9}$$

$$\frac{A: \mathcal{U} \quad x: A \quad xs: \mathtt{Stream} \ A}{\mathtt{cons} \ x \ xs: \mathtt{Stream} \ A} \ \mathtt{I}_{\mathtt{cons}} \tag{2.10}$$

2.2 ITrees as M-types

We want the following rules for ITrees

$$\frac{r:R}{\text{Ret }r:\text{itree }E\ R}\ \text{I}_{\text{Ret}} \tag{2.11}$$

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

Elimination rules

$$\frac{t: \text{itree } \underline{E} \ R}{\text{Tau } t: \text{itree } \underline{E} \ R} \ \mathbf{E}_{\text{Tau}}. \tag{2.13}$$

2.2.1 Delay Monad

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

$$P_{S}(X) = \sum_{x:1+R} \lambda \{ \text{inl } _ \to 1; \text{inr } _ \to 0 \} \ x \to X$$
 (2.14)

This type is equal to the type:

$$P_{\mathbf{S}}(X) = X + R \times (\mathbf{0} \to X) \tag{2.15}$$

we know that $0 \to X \equiv 1$, so we can further reduce to

$$P_{\mathbf{S}}(X) = X + R \tag{2.16}$$

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

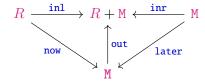


Figure 2.2: Delay monad

now and later using $in = out^{-1}$ together with the injections in and in.

(Later = Tau, Ret = Now)

2.2.2 Tree

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

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

This will give us the polynomial functor:

$$P_{\mathbf{S}}(X) = \sum_{x:R+\sum_{A:U}E} \lambda \{ \text{inl } _ \to \mathbf{0} ; \text{ inr } (A,e) \to A \} x \to X$$
 (2.18)

which simplifies to

$$P_{\mathbf{S}}(X) = (R \times (\mathbf{0} \to X)) + (\sum_{A \neq A} E \ A \times (A \to X)) \tag{2.19}$$

and further

$$P_{\mathbf{S}}(X) = R + \sum_{A:\mathcal{U}} E \ A \times (A \to X) \tag{2.20}$$

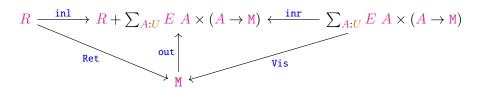


Figure 2.3: TODO: ???

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 M-types. We define ITrees by the container:

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

Then the (reduced) polynomial functor becomes

$$P_{\mathbf{S}}(X) = X + R + \sum_{A:\mathcal{U}} ((E \ A) \times (A \to X)) \tag{2.22}$$

Giving us the diagram

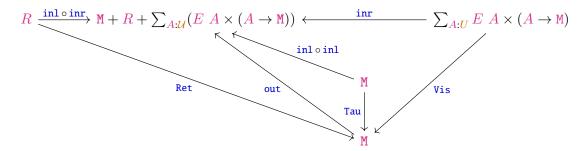


Figure 2.4: TODO: ???

2.3 Co-induction Principle for M-types

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

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

Furthermore for any bisimulation over a final P-coalgebra M-out : $Coalg_S$ we have the following diagram,

$$\operatorname{M-out} \xleftarrow{\pi_1^{\overline{\mathcal{R}}}} \overline{\mathcal{R}} - \alpha_{\overline{\mathcal{R}}} \xrightarrow{\pi_2^{\overline{\mathcal{R}}}} \operatorname{M-out}$$

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

$$R \equiv \equiv \tag{2.23}$$

meaning we can use the relation R, to show that two things of an 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

$$p:R\to \equiv \tag{2.24}$$

$$q: \equiv \to R \tag{2.25}$$

and relations

$$\alpha: p \circ q \equiv \mathsf{id}_{\equiv} \tag{2.26}$$

$$\beta: q \circ p \equiv \mathrm{id}_R \tag{2.27}$$

Complete the construction of equality from any bisimulation relation over an 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.

$$\frac{a, b : R}{\text{Ret } a \cong \text{Ret } b} \text{ EqRet}$$

$$(2.28)$$

$$\frac{t,u: \mathtt{itree} \ E \ R \quad t \cong u}{\mathtt{Tau} \ t \cong \mathtt{Tau} \ u} \ \mathtt{EqTau} \tag{2.29}$$

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

$$(2.30)$$

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

define the $\alpha_{\overline{R}}$ 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 M-type (which is a final coalgebra)



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 M-types preserves the H-level, we can use set-truncated quotients, to define quotient 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

Additions to the Cubical Agda Library

3.1 Lemma 10

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

$$U \equiv \sum_{f:C \to \mathcal{L}} \text{out} \circ f \equiv \text{step } f \tag{3.1}$$

$$\equiv \sum_{f:C\to\mathcal{L}} \operatorname{in}\circ\operatorname{out}\circ f \equiv \operatorname{in}\circ\operatorname{step} f \tag{3.2}$$

$$\equiv \sum_{f:C\to\mathcal{L}} f \equiv \text{in} \circ \text{step } f \tag{3.3}$$

$$\equiv \sum_{f:C\to\mathcal{L}} f \equiv \Psi f \tag{3.4}$$

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

$$\equiv \sum_{c \in Come} e \ c \equiv e \ (\phi \ c) \tag{3.6}$$

$$\equiv \sum_{c:Cone} c \equiv \phi \ c \tag{3.7}$$

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

$$\equiv \sum_{(u,q):Cone} \sum_{p:u \equiv \phi_0} q \equiv_{\lambda i,Cone_1(p\ i)} \phi_1 \ u \ q \tag{3.9}$$

$$\equiv \sum_{(u,p): \sum_{u:Cone_0} u \equiv \phi_0} \sum_{u \neq Cone_1} q \equiv_{\lambda i,Cone_1(p \ i)} \phi_1 \ u \ q \tag{3.10}$$

$$\equiv \sum_{q:Cone_1u_0} q \equiv_{\lambda i,Cone_1(funExt\ p_0\ i)} \phi_1\ u_0\ q \tag{3.11}$$

$$\vdots (3.12)$$

$$\equiv 1 \tag{3.13}$$

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