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

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract

in English…

Resumé

in Danish. . .

Acknowledgments

...

Lasse Letager Hansen,
Aarhus, March 8, 2020.

Contents

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 $S = (A, B)$ 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

$$\begin{gathered} P_S : \mathcal{U} \to \mathcal{U} \\ P(X) := P_S(X) = \sum_{a:A} B(a) \to X \end{gathered} \tag{2.1}$$

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

$$\begin{gathered} Pf : PX \to PY \\ Pf(a, g) = (a, f \circ g) \end{gathered} \tag{2.2}$$

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

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

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

$$P_S(X) = A \times X \tag{2.4}$$

Now we define the coalgebra for this functor with type

$$\texttt{Coalg}_S = \sum_{C:\mathcal{U}} C \to PC \tag{2.5}$$

and morphisms

$$\begin{gathered} \_ \Rightarrow \_ : \texttt{Coalg}_S \to \texttt{Coalg}_S \\ (C, \gamma) \Rightarrow (D, \delta) = \sum_{f:C \to D} \delta \circ f = Pf \circ \gamma \end{gathered} \tag{2.6}$$

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

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

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

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

$$\begin{tikzcd}
A & A \times \texttt{M} \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & \texttt{M} \\
& \texttt{M} \arrow[ul, "\texttt{hd}"] \arrow[u, "\texttt{out}"] \arrow[ur, "\texttt{tl}"'] &
\end{tikzcd}$$

Figure 2.1: M-types of streams

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

$$\frac{A : \mathcal{U} \quad s : \texttt{Stream}\ A}{\texttt{hd}\ s : A}\ \mathrm{E}_{\texttt{hd}} \tag{2.8}$$

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

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

2.2 ITrees as M-types

We want the following rules for ITrees

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

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

Elimination rules

$$\frac{t : \texttt{itree}\ E\ R}{\texttt{Tau}\ t : \texttt{itree}\ E\ R}\ \mathrm{E}_{\texttt{Tau}}. \tag{2.13}$$

2.2.1 Delay Monad

check this statement

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

$$P_S(X) = \sum_{x:\mathbf{1}+R} \lambda\{\texttt{inl}\ \_ \to \mathbf{1}; \texttt{inr}\ \_ \to \mathbf{0}\}\ x \to X \tag{2.14}$$

This type is equal to the type:

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

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

$$P_S(X) = X + R \tag{2.16}$$

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

$$\begin{tikzcd}
R \arrow[r, "\mathtt{inl}"] \arrow[dr, "\mathtt{now}"'] & R + \mathtt{M} & \mathtt{M} \arrow[l, "\mathtt{inr}"'] \arrow[dl, "\mathtt{later}"] \\
& \mathtt{M} \arrow[u, "\mathtt{out}"'] &
\end{tikzcd}$$

Figure 2.2: Delay monad

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

(Later = Tau, Ret = Now)

2.2.2 Tree

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

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

This will give us the polynomial functor:

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

which simplifies to

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

and further

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

$$\begin{tikzcd}
R \arrow[r, "\mathtt{inl}"] \arrow[dr, "\mathtt{Ret}"'] & R + \sum_{A:U} E\ A \times (A \to \mathtt{M}) & \sum_{A:U} E\ A \times (A \to \mathtt{M}) \arrow[l, "\mathtt{inr}"'] \arrow[dl, "\mathtt{Vis}"] \\
& \mathtt{M} \arrow[u, "\mathtt{out}"] &
\end{tikzcd}$$

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 $\texttt{M}$-types. We define ITrees by the container:

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

Then the (reduced) polynomial functor becomes

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

Giving us the diagram

$$\begin{tikzcd}
R \arrow[r, "\texttt{inl} \circ \texttt{inr}"] \arrow[ddrr, "\texttt{Ret}"'] & \texttt{M} + R + \sum_{A:\mathcal{U}} (E\ A \times (A \to \texttt{M})) & & \sum_{A:U} E\ A \times (A \to \texttt{M}) \arrow[ll, "\texttt{inr}"'] \arrow[ddl, "\texttt{Vis}"] \\
& & \texttt{M} \arrow[ul, "\texttt{inl} \circ \texttt{inl}"'] \arrow[d, "\texttt{Tau}"'] & \\
& & \texttt{M} \arrow[uul, "\texttt{out}"] &
\end{tikzcd}$$

Figure 2.4: TODO: ???

2.3 Co-induction Principle for M-types

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

$$\begin{tikzcd}
C\text{-}\gamma & \overline{\mathcal{R}}\text{-}\alpha_{\overline{\mathcal{R}}} \arrow[l, Rightarrow, "{\pi_1}^{\overline{\mathcal{R}}}"'] \arrow[r, Rightarrow, "{\pi_2}^{\overline{\mathcal{R}}}"] & C\text{-}\gamma
\end{tikzcd}$$

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

$$\begin{tikzcd}
\texttt{M}\text{-}\texttt{out} & \overline{\mathcal{R}}\text{-}\alpha_{\overline{\mathcal{R}}} \arrow[l, Rightarrow, "{\pi_1}^{\overline{\mathcal{R}}}"'] \arrow[r, Rightarrow, "{\pi_2}^{\overline{\mathcal{R}}}"] & \texttt{M}\text{-}\texttt{out}
\end{tikzcd}$$

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 $\texttt{M}$-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

$$\alpha : p \circ q \equiv \texttt{id}_{\equiv} \tag{2.26}$$
$$\beta : q \circ p \equiv \texttt{id}_{R} \tag{2.27}$$

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.

$$\frac{a, b : R \quad a \equiv_R b}{\texttt{Ret}\ a \cong \texttt{Ret}\ b}\ \texttt{EqRet} \tag{2.28}$$

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

$$\frac{A : \mathcal{U} \quad e : E\ A \quad k_1, k_2 : A \to \texttt{itree}\ E\ R \quad t \cong u}{\texttt{Vis}\ e\ k_1 \cong \texttt{Tau}\ e\ k_2}\ \texttt{EqVis} \tag{2.30}$$

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 $\alpha_{\overline{R}}$ function

2.4 Examples of fixed points

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

$$\begin{tikzcd}
\texttt{spin} \arrow[d, "\texttt{later}"] \\
\texttt{later spin}
\end{tikzcd}$$

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

Chapter 3

Additions to the Cubical Agda Library

3.1 Lemma 10

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

$$
\begin{aligned}
U &\equiv \sum_{f:C\to\mathcal{L}} \texttt{out} \circ f \equiv \texttt{step}\ f && (3.1)\\
&\equiv \sum_{f:C\to\mathcal{L}} \texttt{in} \circ \texttt{out} \circ f \equiv \texttt{in} \circ \texttt{step}\ f && (3.2)\\
&\equiv \sum_{f:C\to\mathcal{L}} f \equiv \texttt{in} \circ \texttt{step}\ f && (3.3)\\
&\equiv \sum_{f:C\to\mathcal{L}} f \equiv \Psi\ f && (3.4)\\
&\equiv \sum_{c:Cone} e\ c \equiv \Psi\ (e\ c) && (3.5)\\
&\equiv \sum_{c:Cone} e\ c \equiv e\ (\phi\ c) && (3.6)\\
&\equiv \sum_{c:Cone} c \equiv \phi\ c && (3.7)\\
&\equiv \sum_{(u,q):Cone} (u, q) \equiv (\phi_0\ u, \phi_1\ u\ q) && (3.8)\\
&\equiv \sum_{(u,q):Cone} \sum_{p:u\equiv\phi_0\ u} q \equiv_{\lambda i, Cone_1 (p\ i)} \phi_1\ u\ q && (3.9)\\
&\equiv \sum_{(u,p):\sum_{u:Cone_0} u\equiv\phi_0\ u} \sum_{q:Cone_1\ u} q \equiv_{\lambda i, Cone_1 (p\ i)} \phi_1\ u\ q && (3.10)\\
&\equiv \sum_{q:Cone_1 u_0} q \equiv_{\lambda i, Cone_1 (funExt\ p_0\ i)} \phi_1\ u_0\ q && (3.11)\\
&\vdots && (3.12)\\
&\equiv \mathbf{1} && (3.13)
\end{aligned}
$$

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

. . .