M-types and Coinduction in HoTT and Cubical Type Theory

Lasse Letager Hansen, 201912345

Master's Thesis, Computer Science
March 23, 2020
Advisor: Bas Spitters

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract

in English. . .

Resumé

in Danish...

Acknowledgments

Lasse Letager Hansen,
Aarhus, March 23, 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

$$\begin{aligned} \mathtt{P}_S &: \mathcal{U} \to \mathcal{U} \\ \mathtt{P}(X) := \mathtt{P}_S(X) &= \sum_{a:A} B(a) \to X \end{aligned} \tag{2.1}$$

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

$$\begin{aligned} \mathtt{P}f &: \mathtt{P}X \to \mathtt{P}Y \\ \mathtt{P}f(a, g) &= (a, f \circ g) \end{aligned} \tag{2.2}$$

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

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

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

$$\mathtt{P}_S(X) = A \times X \tag{2.4}$$

Now we define the coalgebra for this functor with type

$$\mathtt{Coalg}_S = \sum_{C:\mathcal{U}} C \to \mathtt{P}C \tag{2.5}$$

we denote a coalgera of C and γ as C-γ. The coalgebra morphisms are defined as

$$\begin{aligned} \_\Rightarrow\_ &: \mathtt{Coalg}_S \to \mathtt{Coalg}_S \\ C\text{-}\gamma \Rightarrow D\text{-}\delta &= \sum_{f:C\to D} \delta \circ f = \mathtt{P}f \circ \gamma \end{aligned} \tag{2.6}$$

M-types can now be defined from a container S as the type $\mathtt{M}_S$ such that the coalgebra for $\mathtt{M}_S$ and $\mathtt{out} : \mathtt{M}_S \to \mathtt{P}_S(\mathtt{M}_S)$ fulfills the property

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

that is $\prod_{(C\text{-}\gamma : \mathtt{Coalg}_S)} \mathtt{isContr}(C\text{-}\gamma \Rightarrow \mathtt{M}_S\text{-}\mathtt{out})$. We denote this construction of the M-type as $\mathtt{M}_{(A\,,\,B)}$ or $\mathtt{M}_S$ or just $\mathtt{M}$ when the Container is clear from the context.

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

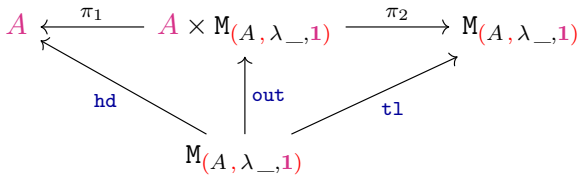

Figure 2.1: M-types of streams

the coalgebra), with inverse $\mathtt{in} : \mathtt{P}_S(\mathtt{M}) \to \mathtt{M}$. We now have a semantic for the rules we would expect for streams, if we let $\mathtt{cons} = \mathtt{in}$ and $\mathtt{stream}\ A = \mathtt{M}_{(A\,,\,\lambda\_,\mathbf{1})}$,

$$\frac{A : \mathcal{U} \quad s : \mathtt{stream}\ A}{\mathtt{hd}\ s : A}\ \mathtt{E}_{\mathtt{hd}} \tag{2.8}$$

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

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

2.2 Closure properties of M-types

We define the product of two containers

$$(A\,,B) \times (C\,,D) \equiv (A \times C\,, \lambda\,(a,c), \mathtt{B}\ a \times \mathtt{D}\ c), \tag{2.11}$$

we can lift this rule, through the diagram in Figure 2.2, used to define M-types. We now prove that

$$\mathtt{P}_{(A\,,\,B)}{}^{n}\ \mathbf{1} \times \mathtt{P}_{(C\,,\,D)}{}^{n}\ \mathbf{1} \equiv \mathtt{P}_{(A\,,\,B)\times(C\,,\,D)}{}^{n}\ \mathbf{1}, \tag{2.12}$$

by induction on n. For $n = 0$, we have $\mathbf{1} \times \mathbf{1} \equiv \mathbf{1}$. For $n = m + 1$, we may assume

$$\mathtt{P}_{(A\,,\,B)}{}^{m}\ \mathbf{1} \times \mathtt{P}_{(C\,,\,D)}{}^{m}\ \mathbf{1} \equiv \mathtt{P}_{(A\,,\,B)\times(C\,,\,D)}{}^{m}\ \mathbf{1}, \tag{2.13}$$

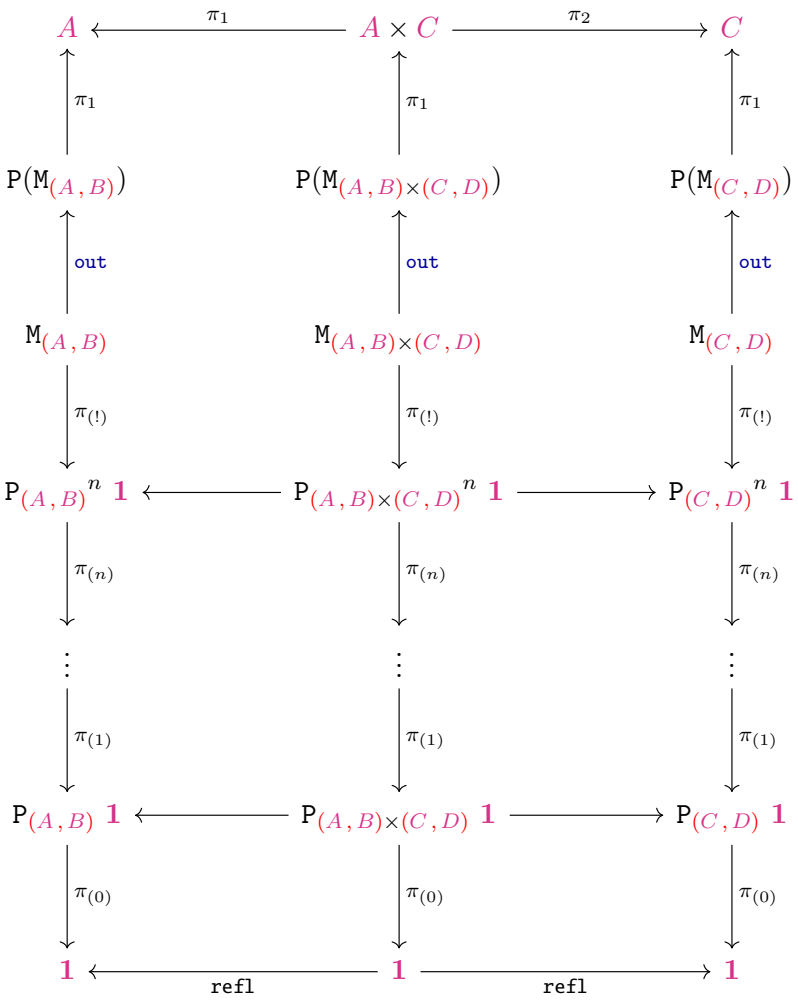

Figure 2.2: TODO

in the following

$$\mathtt{P}_{(A\,,\,B)}{}^{m+1}\ \mathbf{1} \times \mathtt{P}_{(C\,,\,D)}{}^{m+1}\ \mathbf{1} \tag{2.14}$$

$$\equiv \mathtt{P}_{(A\,,\,B)}(\mathtt{P}_{(A\,,\,B)}{}^{m}\ \mathbf{1}) \times \mathtt{P}_{(C\,,\,D)}(\mathtt{P}_{(C\,,\,D)}{}^{m}\ \mathbf{1}) \tag{2.15}$$

$$\equiv \sum_{a:A} B\ a \to \mathtt{P}_{(A\,,\,B)}{}^{m}\ \mathbf{1} \times \sum_{c:C} D\ c \to \mathtt{P}_{(C\,,\,D)}{}^{m}\ \mathbf{1} \tag{2.16}$$

$$\equiv \sum_{a,c:A\times C} (B\ a \to \mathtt{P}_{(A\,,\,B)}{}^{m}\ \mathbf{1}) \times (D\ c \to \mathtt{P}_{(C\,,\,D)}{}^{m}\ \mathbf{1}) \tag{2.17}$$

$$\equiv \sum_{a,c:A\times C} B\ a \times D\ c \to \mathtt{P}_{(A\,,\,B)}{}^{m}\ \mathbf{1} \times \mathtt{P}_{(C\,,\,D)}{}^{m}\ \mathbf{1} \tag{2.18}$$

$$\equiv \sum_{a,c:A\times C} B\ a \times D\ c \to \mathtt{P}_{(A\,,\,B)\times(C\,,\,D)}{}^{m}\ \mathbf{1} \tag{2.19}$$

$$\equiv \mathtt{P}_{(A\,,\,B)\times(C\,,\,D)}(\mathtt{P}_{(A\,,\,B)\times(C\,,\,D)}{}^{m}\ \mathbf{1}) \tag{2.20}$$

$$\equiv \mathtt{P}_{(A\,,\,B)\times(C\,,\,D)}{}^{m+1}\ \mathbf{1} \tag{2.21}$$

taking the limit we get

$$\mathtt{M}_{(A\,,\,B)} \times \mathtt{M}_{(C\,,\,D)} \equiv \mathtt{M}_{(A\,,\,B)\times(C\,,\,D)} \tag{2.22}$$

as an example hereof lets look at streams, where we actually get

$$\mathtt{stream}\ A \times \mathtt{stream}\ B \equiv \mathtt{M}_{(A\,,\lambda\ \_,\mathbf{1})} \times \mathtt{M}_{(B\,,\lambda\ \_,\mathbf{1})} \equiv \mathtt{M}_{(A\,,\lambda\ \_,\mathbf{1})\times(B\,,\lambda\ \_,\mathbf{1})} \equiv \mathtt{stream}\ (A \times B) \tag{2.23}$$

as expected, transporting along equality (2.23) gives us the definition for `zip`.

2.3 Co-induction Principle for M-types

We can now construct a bisimulation: forall coalgebras $C\text{-}\gamma : \mathtt{Coalg}_S$, 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 $\alpha_{\overline{\mathcal{R}}} : \overline{\mathcal{R}} \to \mathtt{P}(\overline{\mathcal{R}})$ forms a P-coalgebra $\overline{\mathcal{R}}\text{-}\alpha_{\overline{\mathcal{R}}} : \mathtt{Coalg}_S$, making the diagram in Figure 2.3 commute (where $\Longrightarrow$ are P-coalgebra morphisms). Furthermore for any bisimulation over a final P-coalgebra $\mathtt{M\text{-}out} : \mathtt{Coalg}_S$

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

Figure 2.3: Bisimulation for a coalgebra

we get the diagram in Figure 2.4, where $\pi_1^{\overline{\mathcal{R}}} = ! = \pi_2^{\overline{\mathcal{R}}}$, which means given $r : \mathcal{R}(m, m')$ we get

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

Figure 2.4: Bisimulation principle for final coalgebra

$m = \pi_1^{\overline{\mathcal{R}}}(m, m', r) = \pi_2^{\overline{\mathcal{R}}}(m, m', r) = m'$.

Better introduction to this proof!

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

$$R \equiv \equiv \tag{2.24}$$

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

$$p : R \to \equiv \tag{2.25}$$
$$q : \equiv \to R \tag{2.26}$$

and relations

$$\alpha : p \circ q \equiv \mathtt{id}_{\equiv} \tag{2.27}$$
$$\beta : q \circ p \equiv \mathtt{id}_{R} \tag{2.28}$$

Complete the construction of equality from any bisimulation relation over an M-type

2.4 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

Quotients of the delay monad

2.5 TODO

- Resumption Monad transformer
- coinduction in Coq is broken
- bisim $\Rightarrow$ eq
- copattern matching
- cubical agda. Relation between M-types defined by coinduction/copattern matching and contructed from W-types
- In agda, coinductive types are defined using Record types, which are Sigma-types.
- In cubical agda, 3.2.2 the issue of productivity is discussed. This can probably be made precise using guarded types.
- streams defined by guarded recursion vs coinduction in guarded cubical agda.
- p3 of the guarded cubical agda paper describes how semantic productivity improves over syntactic productivity
- Reduction of co-inductive types in Coq/agda to (indexed) M-types. Like reduction of strictly positive inductive types to W-types. https://ncatlab.org/nlab/show/W-type
- QIITs have been formalized in agda using private types. Can this also be done in cubical agda (ie without cheating).
- Show that this is the final (quotiented) coalgebra. Does this generalize to QM-types, and what are those constructively ??

Chapter 3

Examples of M-types

3.1 TODO: Place these subsections

3.1.1 Bisimulation of Streams

TODO

3.1.2 Bisimulation of Delay monad

TODO

3.1.3 Bisimulation of ITrees

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

$$\frac{a, b : R \quad a \equiv_R b}{\texttt{Ret}\ a \cong \texttt{Ret}\ b}\ \texttt{EqRet} \tag{3.1}$$

$$\frac{t, u : \texttt{itree E}\ R \quad t \cong u}{\texttt{Tau}\ t \cong \texttt{Tau}\ u}\ \texttt{EqTau} \tag{3.2}$$

$$\frac{A : \mathcal{U} \quad e : \texttt{E}\ A \quad k_1, k_2 : A \to \texttt{itree E}\ R \quad t \cong u}{\texttt{Vis}\ e\ k_1 \cong \texttt{Tau}\ e\ k_2}\ \texttt{EqVis} \tag{3.3}$$

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

3.1.4 Zip Function

We want the diagram in Figure 3.1 to commute, meaning we get the computation rules

$$(\texttt{hd} \times \texttt{hd}) \equiv \texttt{hd} \circ \texttt{zip} \tag{3.4}$$

$$\texttt{zip} \circ (\texttt{tl} \times \texttt{tl}) \equiv \texttt{tl} \circ \texttt{zip} \tag{3.5}$$

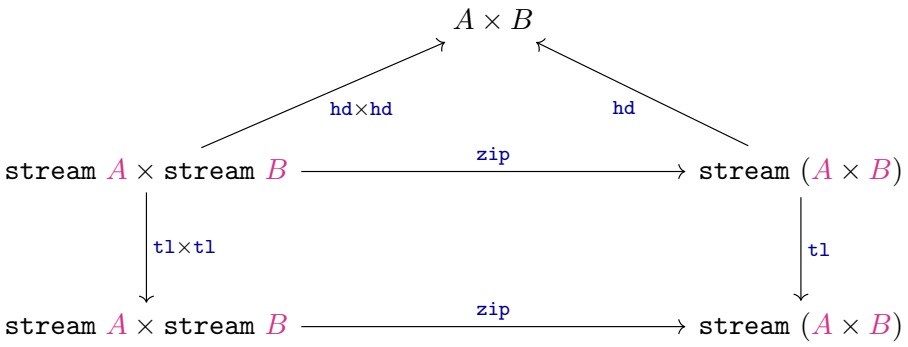

Figure 3.1: TODO

we can define the zip function as we did in the end of the last section. Another way to define the zip function is more directly, using the following lifting property of M-types

$$\begin{aligned}\mathtt{lift_M}\ \left(x : \prod_{n:\mathbb{N}} (A \to \mathtt{P}_S{}^n\ \mathbf{1})\right)\ \left(u : \prod_{n:\mathbb{N}} (A \to \pi_n(x_{n+1}a) \equiv x_n a)\right)\ (a : A) : \mathtt{M}\ S :=\\ (\lambda\, n, x\ n\ a), (\lambda\, n\ i, p\ n\ a\ i).\end{aligned} \tag{3.6}$$

To use this definition, we first define some helper functions

$$\mathtt{zip}_X\ n\ (x, y) = \begin{cases} \mathbf{1} & \mathtt{if}\ n = 0 \\ (\mathtt{hd}\ x, \mathtt{hd}\ y), (\lambda\, \_, \mathtt{zip}_X\ m\ (\mathtt{tl}\ x, \mathtt{tl}\ y)), & \mathtt{if}\ n = m + 1 \end{cases} \tag{3.7}$$

$$\mathtt{zip}_\pi\ n\ (x, y) = \begin{cases} \mathtt{refl} & \mathtt{if}\ n = 0 \\ \lambda\, i, (\mathtt{hd}\ x, \mathtt{hd}\ y), (\lambda\, \_, \mathtt{zip}_\pi\ m\ (\mathtt{tl}\ x, \mathtt{tl}\ y)\ i), & \mathtt{if}\ n = m + 1 \end{cases}, \tag{3.8}$$

we can then define

$$\mathtt{zip}_{lift}\ (x, y) := \mathtt{lift_M}\ \mathtt{zip}_X\ \mathtt{zip}\ (x, y). \tag{3.9}$$

Equality of Zip Definitions

We would expect that the two definitions for zip are equal

$$\begin{aligned} \mathtt{transport}_?\ a &\equiv \mathtt{zip}_{lift}\ a && (3.10)\\ &\equiv \mathtt{lift_M}\ \mathtt{zip}_X\ \mathtt{zip}_\pi\ (x, y) && (3.11)\\ &\equiv (\lambda\, n, \mathtt{zip}_X\ n\ (x, y)), (\lambda\, n\ i, \mathtt{zip}_\pi\ n\ (x, y)\ i) && (3.12)\end{aligned}$$

zero case X

$$\mathtt{zip}_X\ 0\ (x, y) \equiv \mathbf{1} \tag{3.13}$$

Sucessor case X

$$\begin{aligned} \mathtt{zip}_X\ (m + 1)\ (x, y) &\equiv (\mathtt{hd}\ x, \mathtt{hd}\ y), (\lambda\, \_, \mathtt{zip}_X\ m\ (\mathtt{tl}\ x, \mathtt{tl}\ y)) && (3.14)\\ &\equiv (\mathtt{hd}\ x, \mathtt{hd}\ y), (\lambda\, \_, ?\ (\mathtt{tl}\ a)) && (3.15)\\ &\equiv (\mathtt{hd}\ (\mathtt{transport}_? a)), (\lambda\, \_, \mathtt{transport}_? (\mathtt{tl}\ a)) && (3.16)\\ &\equiv transport_? a && (3.17)\\ & && (3.18)\end{aligned}$$

Zero case π: $(\lambda\, i, \mathtt{zip}_\pi\ 0\ (x, y)\ i \equiv refl)$.

$$\equiv (), (\lambda\, i, \mathtt{zip}_\pi\ 0\ (x, y)\ i) \tag{3.19}$$
$$\equiv \mathbf{1}, \mathtt{refl} \tag{3.20}$$
$$\tag{3.21}$$

successor case

$$\equiv (\mathtt{zip}_X\ (m+1)\ (x, y)), (\lambda\, i, \mathtt{zip}_\pi\ (m+1)\ (x, y)\ i) \tag{3.22}$$
$$\equiv ((\mathtt{hd}\ x, \mathtt{hd}\ y), (\lambda\, \_, \mathtt{zip}_X\ m\ (\mathtt{tl}\ x, \mathtt{tl}\ y))), (\lambda\, i, (\mathtt{hd}\ x, \mathtt{hd}\ y), (\lambda\, \_, \mathtt{zip}_\pi\ m\ (\mathtt{tl}\ x, \mathtt{tl}\ y)\ i)) \tag{3.23}$$

Complete this proof

3.1.5 Examples of Fixed Points

Zeros

Let us try to define the zero stream, we do this by lifting the functions

$$\mathtt{const}_X\ (n : \mathbb{N})\ (c : \mathbb{N}) := \begin{cases} \mathbf{1} & n = 0 \\ (c, \lambda\, \_, \mathtt{const}_X\ m\ c) & n = m + 1 \end{cases} \tag{3.24}$$

$$\mathtt{const}_\pi\ (n : \mathbb{N})\ (c : \mathbb{N}) := \begin{cases} \mathtt{refl} & n = 0 \\ \lambda\, i, (c, \lambda\, \_, \mathtt{const}_\pi\ m\ c\ i) & n = m + 1 \end{cases} \tag{3.25}$$

to get the definition of zero stream

$$\mathtt{zeros} := \mathtt{lift}_\mathtt{M}\ \mathtt{const}_\mathtt{X}\ \mathtt{const}_\pi\ 0. \tag{3.26}$$

We want to show that we get the expected properties, such as

$$\mathtt{hd}\ \mathtt{zeros} \equiv 0 \tag{3.27}$$
$$\mathtt{tl}\ \mathtt{zeros} \equiv \mathtt{zeros} \tag{3.28}$$

Spin

We want to define spin, as being the fixed point $\mathtt{spin} = \mathtt{later}\ \mathtt{spin}$, 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

3.2 Stream Formalization using M-types

As described earlier, given a type A we define the stream of that type as

$$\mathtt{stream}\ A := \mathtt{M}_{(A\,, \lambda\, \_, \mathbf{1})} \tag{3.29}$$

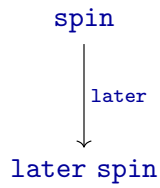

Figure 3.2: TODO

When taking the head of a stream, we get

$$\begin{aligned}
\texttt{hd}\ (\texttt{cons}\ x\ xs) &\equiv \pi_1\ \texttt{out}\ (\texttt{cons}\ x\ xs) && (3.30)\\
&\equiv \pi_1\ \texttt{out}\ (\texttt{in}\ (x, \lambda_, xs)) && (3.31)\\
&\equiv \pi_1\ (x, \lambda_, xs) && (3.32)\\
&\equiv x && (3.33)
\end{aligned}$$

and similarly for the tail of the stream

$$\begin{aligned}
\texttt{tl}\ (\texttt{cons}\ x\ xs) &\equiv \pi_2\ \texttt{out}\ (\texttt{cons}\ x\ xs) && (3.34)\\
&\equiv \pi_2\ \texttt{out}\ (\texttt{in}\ (x, \lambda_, xs)) && (3.35)\\
&\equiv \pi_2\ (x, \lambda_, xs) && (3.36)\\
&\equiv xs && (3.37)
\end{aligned}$$

and the other direction is also true

$$\begin{aligned}
\texttt{cons}(\texttt{hd}\ s, \texttt{tl}\ s) &\equiv \texttt{in}\ (\texttt{hd}\ s, \texttt{tl}\ s) && (3.38)\\
&\equiv \texttt{in}\ (\pi_1\ (\texttt{out}\ s), \pi_2\ (\texttt{out}\ s)) && (3.39)\\
&\equiv \texttt{in}\ (\texttt{out}\ s) && (3.40)\\
&\equiv s. && (3.41)
\end{aligned}$$

When forming elements of the M-type, we want to do it by lifting it though the definition of the M-type, meaning we want to define a function $\texttt{cons}' : (\mathbb{N} \to A) \to \texttt{stream}\ A$ as

$$\texttt{cons}' f = \texttt{lift}_{\texttt{M}}\ (\lambda c\, n, f\ c) \tag{3.42}$$

$$\texttt{cons}' f = \texttt{lift}_{\texttt{M}}\ (\lambda c\, n, f\ c) \tag{3.43}$$

3.3 ITrees as M-types

We want the following rules for ITrees

$$\frac{r : R}{\texttt{Ret}\ r : \texttt{itree}\ \texttt{E}\ R}\ \texttt{I}_{\texttt{Ret}} \tag{3.44}$$

$$\frac{A : \mathcal{U} \quad a : \texttt{E}\ A \quad f : A \to \texttt{itree}\ \texttt{E}\ R}{\texttt{Vis}\ a\ f : \texttt{itree}\ \texttt{E}\ R}\ \texttt{I}_{\texttt{Vis}}. \tag{3.45}$$

Elimination rules

$$\frac{t : \texttt{itree}\ \texttt{E}\ R}{\texttt{Tau}\ t : \texttt{itree}\ \texttt{E}\ R}\ \texttt{E}_{\texttt{Tau}}. \tag{3.46}$$

3.3.1 Delay Monad

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

check this statement

$$\mathtt{P}_S(X) = \sum_{x:\mathbf{1}+R} \lambda\{\texttt{inl}\ \_ \to \mathbf{1}; \texttt{inr}\ \_ \to \mathbf{0}\}\ x \to X, \tag{3.47}$$

which is equal to

$$\mathtt{P}_S(X) = X + R \times (\mathbf{0} \to X). \tag{3.48}$$

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

$$\mathtt{P}_S(X) = X + R \tag{3.49}$$

meaning we get the following diagram.

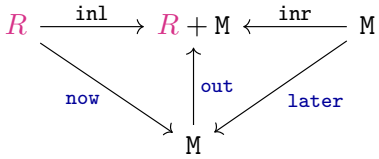

Figure 3.3: Delay monad

Meaning we can define the operations `now` and `later` using $\texttt{in} = \texttt{out}^{-1}$ together with the injections `inl` and `inr`.

(Later = Tau, Ret = Now)

3.3.2 Tree

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

$$S = \left(R + \sum_{A:\mathcal{U}} \mathtt{E}\ A\ ,\ \lambda\{\texttt{inl}\ \_ \to \mathbf{0}\ ;\ \texttt{inr}\ (A, e) \to A\}\right). \tag{3.50}$$

This will give us the polynomial functor

$$\mathtt{P}_S(X) = \sum_{x:R+\sum_{A:\mathcal{U}} \mathtt{E}\ A} \lambda\{\texttt{inl}\ \_ \to \mathbf{0}\ ;\ \texttt{inr}\ (A, e) \to A\}\ x \to X, \tag{3.51}$$

which simplifies to

$$\mathtt{P}_S(X) = (R \times (\mathbf{0} \to X)) + (\sum_{A:\mathcal{U}} \mathtt{E}\ A \times (A \to X)), \tag{3.52}$$

and further

$$\mathtt{P}_S(X) = R + \sum_{A:\mathcal{U}} \mathtt{E}\ A \times (A \to X). \tag{3.53}$$

We get the following diagram for the P-coalgebra.
Again we can define `Ret` and `Vis` using the `in` function.

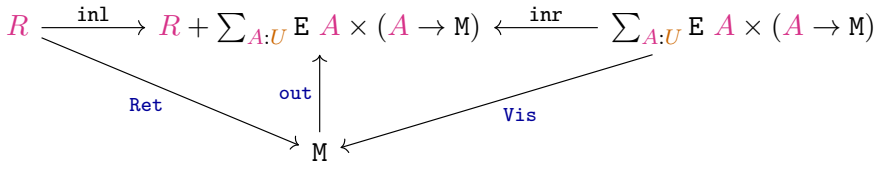

Figure 3.4: TODO

3.3.3 ITrees

Now we should have all the knowledge needed to make ITrees using M-types. We define ITrees by the container

$$S = \left(\mathbf{1} + R + \sum_{A:\mathcal{U}} (\mathtt{E}\ A) \quad , \quad \lambda \{ \mathtt{inl}\ (\mathtt{inl}\ \_) \to \mathbf{1}\ ;\ \mathtt{inl}\ (\mathtt{inr}\ \_) \to \mathbf{0}\ ;\ \mathtt{inr}(A, \_) \to A \} \right). \tag{3.54}$$

Such that the (reduced) polynomial functor becomes

$$\mathtt{P}_S(X) = X + R + \sum_{A:\mathcal{U}} ((\mathtt{E}\ A) \times (A \to X)) \tag{3.55}$$

Giving us the diagram

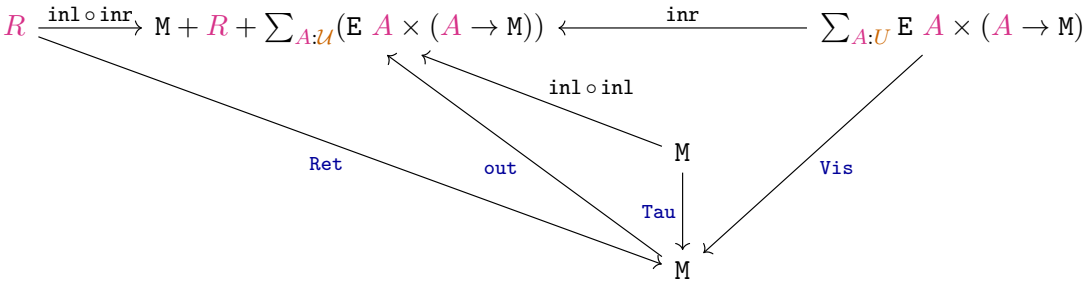

Figure 3.5: TODO

Chapter 4

Additions to the Cubical Agda Library

4.1 Lemma 10

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

$$
\begin{aligned}
U &\equiv \sum_{f:C\to\mathcal{L}} \texttt{out} \circ f \equiv \texttt{step}\ f && (4.1)\\
&\equiv \sum_{f:C\to\mathcal{L}} \texttt{in} \circ \texttt{out} \circ f \equiv \texttt{in} \circ \texttt{step}\ f && (4.2)\\
&\equiv \sum_{f:C\to\mathcal{L}} f \equiv \texttt{in} \circ \texttt{step}\ f && (4.3)\\
&\equiv \sum_{f:C\to\mathcal{L}} f \equiv \Psi\ f && (4.4)\\
&\equiv \sum_{c:Cone} e\ c \equiv \Psi\ (e\ c) && (4.5)\\
&\equiv \sum_{c:Cone} e\ c \equiv e\ (\phi\ c) && (4.6)\\
&\equiv \sum_{c:Cone} c \equiv \phi\ c && (4.7)\\
&\equiv \sum_{(u,q):Cone} (u,q) \equiv (\phi_0\ u, \phi_1\ u\ q) && (4.8)\\
&\equiv \sum_{(u,q):Cone} \sum_{p:u\equiv\phi_0\ u} q \equiv_{\lambda i, Cone_1(p\ i)} \phi_1\ u\ q && (4.9)\\
&\equiv \sum_{(u,p):\sum_{u:Cone_0} u\equiv\phi_0\ u} \sum_{q:Cone_1\ u} q \equiv_{\lambda i, Cone_1(p\ i)} \phi_1\ u\ q && (4.10)\\
&\equiv \sum_{q:Cone_1 u_0} q \equiv_{\lambda i, Cone_1(funExt\ p_0\ i)} \phi_1\ u_0\ q && (4.11)\\
&\vdots && (4.12)\\
&\equiv \mathbf{1} && (4.13)
\end{aligned}
$$

4.2 Lemma 13

$$\sum_{((a,q):\sum_{a:\mathbb{N}\to A}\prod_{n:\mathbb{N}} a_{n+1}\equiv a_n)}\ \sum_{(u:\prod_{n:\mathbb{N}} Ba_n\to X_n)}\ \prod_{(n:\mathbb{N})} \pi\circ u_{n+1} \equiv_{\lambda i,B(q_n\ i)\to X_n} u_n \tag{4.14}$$

$$\equiv \sum_{a:A}\ \sum_{u:\prod_{n:\mathbb{N}} Ba\to X_n}\ \prod_{n:\mathbb{N}} \pi\circ u_{n+1}\equiv u_n \tag{4.15}$$

$$\equiv \sum_{a:A}\ \sum_{u:\prod_{n:\mathbb{N}} Ba\to X_n}\ \prod_{n:\mathbb{N}} \pi\circ u_{n+1}\equiv u_n \tag{4.16}$$

4.3 Missing postulates

Combine

For all $X:\mathbb{N}\to U$ and $p:\prod_{n:\mathbb{N}} X\ (n+1)\to X\ n\to U$

$$\sum_{x_0:X_0}\ \sum_{y:\prod_{n:\mathbb{N}} X_{n+1}} (p\ y_0\ x_0)\times\left(\prod_{n:\mathbb{N}} p\ y_{n+1}\ y_n\right) \tag{4.17}$$

$$\equiv \sum_{x_0:X_0}\ \sum_{y:\prod_{n:\mathbb{N}} X_{n+1}} (p\ y_0\ x_0)\times\left(\prod_{n:\mathbb{N}} p\ y_{n+1}\ y_n\right) \tag{4.18}$$

$$\equiv \sum_{x:\prod_{n:\mathbb{N}}\to X_n} (p\ x_1\ x_0)\times\left(\prod_{n:\mathbb{N}} p\ x_{n+2}\ x_{n+1}\right) \tag{4.19}$$

Chapter 5

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