M-types and Coinduction in HoTT and Cubical Type Theory

Lasse Letager Hansen, 201912345

Master's Thesis, Computer Science
April 14, 2020
Advisor: Bas Spitters

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract

in English. . .

Resumé

in Danish...

Acknowledgments

Lasse Letager Hansen,
Aarhus, April 14, 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

Definition 2.1.1. 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

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

Example 2.1.1. The type for streams over the type A is 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, so this will be equivalent to the definition

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

Definition 2.1.2. The coalgebra for the functor $\mathtt{P}$ is defined as

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

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

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

Definition 2.1.3. 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 M when the Container is clear from the context.

Example 2.1.2. 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 out is an isomorphism (because

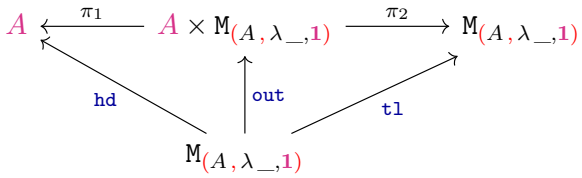

Figure 2.1: M-types of streams

of the finality of 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 $\mathtt{stream}\ A = \mathtt{M}_{(A\,,\,\lambda\_,\mathbf{1})}$,

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

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

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

Lemma 2.1.1. *Given* $\ell : \prod_{(n:\mathbb{N})}(X_n \to X_{n+1})$ *and* $y : \sum_{x:\prod_{(n:\mathbb{N})} X_n} x_{n+1} \equiv l_n(x_n)$ *we can collapse the chain* $\mathcal{L} \equiv X_0$.

Proof. We define this collapse by the equivalence

$$\mathtt{fun}_{\mathcal{L}collapse}\ (x, r) = x_0 \tag{2.11}$$

$$\mathtt{inv}_{\mathcal{L}collapse}\ x_0 = (\lambda n, \mathtt{transport}\ \ell^n\ x_0), (\lambda\, n, \mathtt{refl}_{\mathtt{transport}\ \ell^{(n+1)}\ x_0}) \tag{2.12}$$

$$\mathtt{rinv}\ x_0 = \mathtt{refl}_{x_0} \tag{2.13}$$

For $\mathtt{linv}\ (x, r)$ we define a fiber (X, z, l) over $\mathbb{N}$ given some $z : X_0$. Then any element of the type $\sum_{x:\prod_{(n:\mathbb{N})} X_n} x_{n+1} \equiv l_n(x_n)$ is equal to a section over this fiber, specifically y is equal to a fiber. Since this section is defined over $\mathbb{N}$ which is an initial algebra for the functor $GY = \mathbf{1} + Y$, we get that the section are contractible, meaning $y \equiv \mathtt{inv}_{\mathcal{L}collapse}(\mathtt{fun}_{\mathcal{L}collapse}\ y)$, since both are equal to such sections. □

Theorem 2.1.2. *We want to proof the construction of* in *and* out *for the container* $(A\,,B)$ *more precisely we want to define the equality*

$$shift : \mathcal{L} \equiv P\mathcal{L} \tag{2.14}$$

where $P\mathcal{L}$ *is the limit of a shifted sequence.*

Proof. We do this we the two helper lemmas

$$\alpha : \mathcal{L}^P \equiv P\mathcal{L} \tag{2.15}$$

$$\mathcal{L}unique : \mathcal{L} \equiv \mathcal{L}^P \tag{2.16}$$

We can define $\mathcal{L}unique$ by the equivalence

$$\texttt{fun}_{\mathcal{L}unique}\ (a,b) = \left(\lambda\, n, \begin{cases} \texttt{tt} & n = 0 \\ a\ m & n = m+1 \end{cases}\right), \left(\lambda\, n \begin{cases} \texttt{refl}_{\texttt{tt}} & n = 0 \\ b\ m & n = m+1 \end{cases}\right) \tag{2.17}$$

$$\texttt{inv}_{\mathcal{L}unique}\ (a,b) = a \circ \texttt{incr}, b \circ \texttt{incr} \tag{2.18}$$

$$\texttt{rinv}_{\mathcal{L}unique}\ (a,b) = \texttt{refl} \tag{2.19}$$

$$\texttt{linv}_{Lunique}\ (a,b) = \texttt{refl} \tag{2.20}$$

The definition of α follows from Lemma 2.1.1,

$$\mathcal{L}^P \equiv \sum_{(u:\mathcal{L}_{(\lambda\_,A,\texttt{id})})} \prod_{(n:\mathbb{N})} \pi_{(n)} \circ u_{n+1} \equiv_{\lambda\, x, B(\pi_2\, a\, n\, x) \to X_n} u_n \tag{2.21}$$

$$\equiv \sum_{(a:A)} \sum_{(u:\prod_{(n:\mathbb{N})} Ba \to X_n)} \prod_{(n:\mathbb{N})} \pi_{(n)} \circ u_{n+1} \equiv u_n \tag{2.22}$$

$$\equiv P\mathcal{L} \tag{2.23}$$

We can then define $shift$ as

$$shift = \texttt{sym}\ \alpha \cdot \mathcal{L}unique \tag{2.24}$$

Then we define $\texttt{in} = \texttt{transport}\ shift$ and $\texttt{out} = \texttt{transport}\ (\texttt{sym}\ shift)$, since $\texttt{in}$ and $\texttt{out}$ are part of an equality relation ($shift$), they become embeddings. □

2.2 Closure properties of M-types

We want to show that M-types are closed under simple operations, we start by looking at the product.

2.2.1 Product of M-types

We start with containers and work up to M-types.

Definition 2.2.1. The product of two containers is defined as

$$(A\,, B) \times (C\,, D) \equiv (A \times C\,, \lambda\,(a, c), \texttt{B}\ a \times \texttt{D}\ c). \tag{2.25}$$

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

Theorem 2.2.1. *For any $n : \mathbb{N}$ the following is true*

$$\texttt{P}_{(A\,,B)}{}^n\ \mathbf{1} \times \texttt{P}_{(C\,,D)}{}^n\ \mathbf{1} \equiv \texttt{P}_{(A\,,B)\times(C\,,D)}{}^n\ \mathbf{1}. \tag{2.26}$$

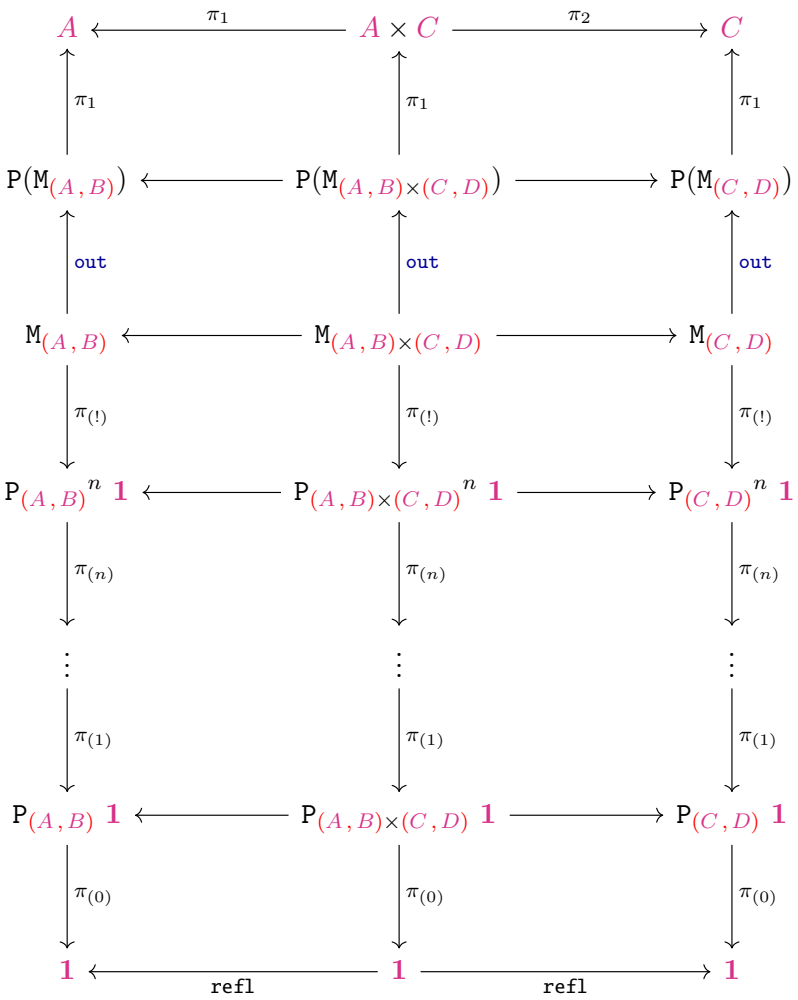

Figure 2.2: Diagram for products of chains

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

$$\mathtt{P}_{(A\,,\,B)}{}^{m}\ \mathbf{1} \times \mathtt{P}_{(C\,,\,D)}{}^{m}\ \mathbf{1} \equiv \mathtt{P}_{(A\,,\,B)\times(C\,,\,D)}{}^{m}\ \mathbf{1}, \tag{2.27}$$

in the following

$$\begin{aligned}
&\mathtt{P}_{(A\,,\,B)}{}^{m+1}\ \mathbf{1} \times \mathtt{P}_{(C\,,\,D)}{}^{m+1}\ \mathbf{1} && (2.28)\\
\equiv\ &\mathtt{P}_{(A\,,\,B)}(\mathtt{P}_{(A\,,\,B)}{}^{m}\ \mathbf{1}) \times \mathtt{P}_{(C\,,\,D)}(\mathtt{P}_{(C\,,\,D)}{}^{m}\ \mathbf{1}) && (2.29)\\
\equiv\ &\sum_{a:A} B\ a \to \mathtt{P}_{(A\,,\,B)}{}^{m}\ \mathbf{1} \times \sum_{c:C} D\ c \to \mathtt{P}_{(C\,,\,D)}{}^{m}\ \mathbf{1} && (2.30)\\
\equiv\ &\sum_{a,c:A\times C} (B\ a \to \mathtt{P}_{(A\,,\,B)}{}^{m}\ \mathbf{1}) \times (D\ c \to \mathtt{P}_{(C\,,\,D)}{}^{m}\ \mathbf{1}) && (2.31)\\
\equiv\ &\sum_{a,c:A\times C} B\ a \times D\ c \to \mathtt{P}_{(A\,,\,B)}{}^{m}\ \mathbf{1} \times \mathtt{P}_{(C\,,\,D)}{}^{m}\ \mathbf{1} && (2.32)
\end{aligned}$$

$$\equiv \sum_{a,c:A\times C} B\ a \times D\ c \to \mathtt{P}_{(A\,,B)\times(C\,,D)}{}^{m}\ \mathbf{1} \tag{2.33}$$

$$\equiv \mathtt{P}_{(A\,,B)\times(C\,,D)}(\mathtt{P}_{(A\,,B)\times(C\,,D)}{}^{m}\ \mathbf{1}) \tag{2.34}$$

$$\equiv \mathtt{P}_{(A\,,B)\times(C\,,D)}{}^{m+1}\ \mathbf{1} \tag{2.35}$$

taking the limit of (2.26) we get

$$\mathtt{M}_{(A\,,B)} \times \mathtt{M}_{(C\,,D)} \equiv \mathtt{M}_{(A\,,B)\times(C\,,D)}. \tag{2.36}$$

□

Example 2.2.1. For streams we get

$$\mathtt{stream}\ A \times \mathtt{stream}\ B \equiv \mathtt{M}_{(A\,,\lambda\ \_,\mathbf{1})} \times \mathtt{M}_{(B\,,\lambda\ \_,\mathbf{1})} \equiv \mathtt{M}_{(A\,,\lambda\ \_,\mathbf{1})\times(B\,,\lambda\ \_,\mathbf{1})} \equiv \mathtt{stream}\ (A \times B) \tag{2.37}$$

as expected. Transporting along (2.37) gives us a definition for `zip`.

2.3 Co-induction Principle for M-types

We can now construct a co-induction principle given a bisimulation relation

Definition 2.3.1. Forall coalgebras $C\text{-}\gamma : \mathtt{Coalg}_S$, given a relation $\mathcal{R} : C \to C \to \mathcal{U}$ and a type $\overline{\mathcal{R}} = \sum_{a:C}\sum_{b:C} a\ \mathcal{R}\ b$, such that $\overline{\mathcal{R}}$ and $\alpha_{\mathcal{R}} : \overline{\mathcal{R}} \to \mathtt{P}(\overline{\mathcal{R}})$ forms a P-coalgebra $\overline{\mathcal{R}}\text{-}\alpha_{\mathcal{R}} : \mathtt{Coalg}_S$, making the diagram in Figure 2.3 commute ($\Longrightarrow$ represents P-coalgebra morphisms).

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

Figure 2.3: Bisimulation for a coalgebra

Definition 2.3.2 (Co-induction principle)**.** Given a relation $\mathcal{R}$, that is part of a bisimulation over a final P-coalgebra $\mathtt{M\text{-}out} : \mathtt{Coalg}_S$ we get the diagram in Figure 2.4,

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

Figure 2.4: Bisimulation principle for final coalgebra

where $\pi_1^{\overline{\mathcal{R}}} = ! = \pi_2^{\overline{\mathcal{R}}}$, which means given $r : m\ \mathcal{R}\ m'$ we get the equation

$$m = \pi_1^{\overline{\mathcal{R}}}(m, m', r) = \pi_2^{\overline{\mathcal{R}}}(m, m', r) = m'. \tag{2.38}$$

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

2.4.1 Quotient inductive-inductive types (QIITs)

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 Q`M`-types, and what are those constructively ??

Chapter 3

Contractive

We want to show that

$$\sum_{(a:A)} \sum_{(u:\prod_{(n:\mathbb{N})} B\ a \to X_n)} \prod_{(n:\mathbb{N})} \pi_{(n)} \circ (u_{n+1}) \equiv u_n \tag{3.1}$$

$$\equiv \sum_{(a:\sum_{(a:(n:\mathbb{N})\to A)} \prod_{(n:\mathbb{N})} a_{n+1} \equiv a_n)} \sum_{(u:(n:\mathbb{N})\to B\ (\pi_1\ a)_n \to X_n)} \prod_{(n:\mathbb{N})} \pi_{(n)} \circ u_{n+1} \equiv_{(\lambda x, B\ ((\pi_2\ a)_n\ x)) \to X_n)} u_n \tag{3.2}$$

we have the two functions

$$\texttt{fun}\ (x, y) := ((\lambda\, n, x), (\lambda\, n\ i, x)), y \tag{3.3}$$

and its inverse (we use the notation $y_{(!n)} = y_n \cdot y_{n-1} \cdot \ \ldots\ \cdot y_1 \cdot y_0$)

$$\texttt{inv}\ ((x, y), (z, w)) := x_0, ((\lambda\, n\ a, z\ n\ (\texttt{subst}\ B\ y_{(!n)}\ a)), (\lambda\, n\ i\ a, w\ n\ i\ (\texttt{subst}\ B\ ?\ a))) \tag{3.4}$$

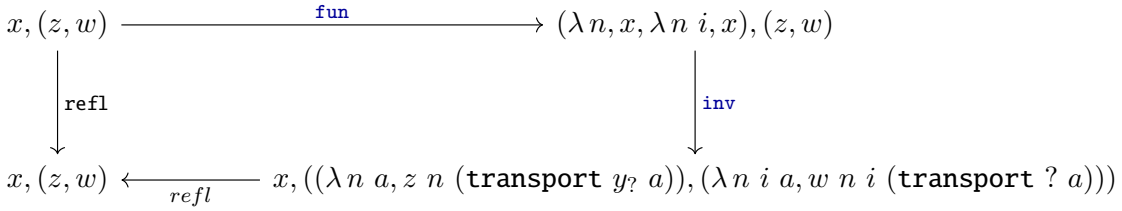

Figure 3.1: Left inverse

so we want

$$\texttt{invhelper0}\ ((\lambda\, n, (x, y), \lambda\, n\ i, (x, y)), (z, w)) \tag{3.5}$$

$$\equiv \lambda n, \begin{cases} \texttt{tt} & n = 0 \end{cases} \tag{3.6}$$

$$\equiv z \tag{3.7}$$

$$\texttt{invhelper1}\ ((\lambda\, n, (x, y), \lambda\, n\ i, (x, y)), (z, w)) \equiv w \tag{3.8}$$

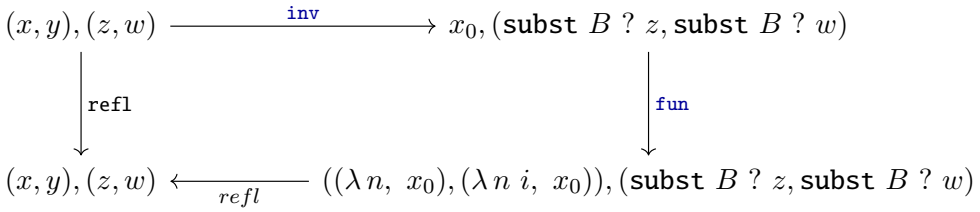

Figure 3.2: Right inverse

Chapter 4

Examples of M-types

4.1 TODO: Place these subsections

What makes a relation a bisimulation? Is bisim and equality equal.

4.1.1 Identity Bisimulation

Lets start with a simple example of a bisimulation namely the one given by the identity relation for any M-type.

Lemma 4.1.1. *The identity relation* $(\cdot \equiv \cdot)$ *is a bisimulation for any final coalgebra* $\mathtt{M}_S\text{-}\mathtt{out}$ *defined over an* M*-type.*

Proof. We first define the function

$$\begin{aligned} \alpha_{\equiv} &: \equiv \to \mathtt{P}(\equiv) \\ \alpha_{\equiv}(x, y) &:= \pi_1\ (\mathtt{out}\ x)\ ,\ (\lambda\, b, (\pi_2\ (\mathtt{out}\ x)\ b\ ,\ \mathtt{refl}_{(\pi_2\ (\mathtt{out}\ x)\ b)})) \end{aligned} \tag{4.1}$$

and the two projections

$$\pi_1^{\equiv} = (\pi_1, \mathtt{funExt}\ \lambda\,(a, b, r), \mathtt{refl}_{\mathtt{out}\,a}) \tag{4.2}$$

$$\pi_2^{\equiv} = (\pi_2, \mathtt{funExt}\ \lambda\,(a, b, r), \mathtt{cong}_{\mathtt{out}}\ (\mathtt{sym}\ r)). \tag{4.3}$$

This defines the bisimulation, given by the diagram in Figure 4.1. □

$$\mathtt{M}\text{-}\mathtt{out} \xleftarrow{\pi_1^{\equiv}} \equiv\text{-}\alpha_{\equiv} \xrightarrow{\pi_2^{\equiv}} \mathtt{M}\text{-}\mathtt{out}$$

Figure 4.1: Identity bisimulation

4.1.2 Bisimulation of Streams

TODO

4.1.3 Bisimulation of Delay Monad

We want to define a strong bisimulation relation $\sim_{\texttt{delay}}$ for the delay monad,

Definition 4.1.1. The relation $\sim_{\texttt{delay}}$ is defined by the following rules

$$\frac{R : U \quad r : R}{\texttt{now}\ r \sim_{\texttt{delay}} \texttt{now}\ r : \mathcal{U}}\ \texttt{now}\sim \tag{4.4}$$

$$\frac{R : \mathcal{U} \quad t : \texttt{delay}\ R \quad u : \texttt{delay}\ R \quad t \sim_{\texttt{delay}} u : \mathcal{U}}{\texttt{later}\ t \sim_{\texttt{delay}} \texttt{later}\ u : \mathcal{U}}\ \texttt{later}\sim \tag{4.5}$$

Theorem 4.1.2. *The relation* $\sim_{\texttt{delay}}$ *is a bisimulation for* $\texttt{delay}\ R$.

Proof. First we define the function

$$\begin{aligned} \alpha_{\sim_{\texttt{delay}}} &: \overline{\sim_{delay}} \to \mathsf{P}(\overline{\sim_{delay}}) \\ \alpha_{\sim_{\texttt{delay}}}\ (a, b, \texttt{now}\sim\ r) &:= (\texttt{inr}\ r, \lambda\,()) \\ \alpha_{\sim_{\texttt{delay}}}\ (a, b, \texttt{later}\sim\ x\ y\ q) &:= (\texttt{inl}\ \texttt{tt}, \lambda\,\_, (x, y, q)) \end{aligned} \tag{4.6}$$

then we define the projections

$$\pi_1^{\overline{\sim_{delay}}} = \left(\pi_1 \quad , \quad \texttt{funExt}\ \lambda\,(a, b, p), \begin{cases} (\texttt{inr}\ r, \lambda\,()) & p = \texttt{now}\sim\ r \\ (\texttt{inl}\ \texttt{tt}, \lambda\,\_, x) & p = \texttt{later}\sim\ x\ y\ q \end{cases}\right) \tag{4.7}$$

$$\pi_2^{\overline{\sim_{delay}}} = \left(\pi_2 \quad , \quad \texttt{funExt}\ \lambda\,(a, b, p), \begin{cases} (\texttt{inr}\ r, \lambda\,()) & p = \texttt{now}\sim\ r \\ (\texttt{inl}\ \texttt{tt}, \lambda\,\_, y) & p = \texttt{later}\sim\ x\ y\ q \end{cases}\right) \tag{4.8}$$

$$\tag{4.9}$$

This defines the bisimulation, given by the diagram in Figure 4.2. □

$$\texttt{delay}\ R\text{-}\texttt{out} \xleftarrow{\pi_1^{\overline{\sim_{delay}}}} \overline{\sim_{\texttt{delay}}}\text{-}\alpha_{\sim_{\texttt{delay}}} \xrightarrow{\pi_2^{\overline{\sim_{delay}}}} \texttt{delay}\ R\text{-}\texttt{out}$$

Figure 4.2: Strong bisimulation for delay monad

4.1.4 Bisimulation of ITrees

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

$$\frac{a, b : R \quad a \equiv_R b}{\texttt{Ret}\ a \cong \texttt{Ret}\ b}\ \texttt{EqRet} \tag{4.10}$$

$$\frac{t, u : \texttt{itree}\ \texttt{E}\ R \quad t \cong u}{\texttt{Tau}\ t \cong \texttt{Tau}\ u}\ \texttt{EqTau} \tag{4.11}$$

$$\frac{A : \mathcal{U} \quad e : \texttt{E}\ A \quad k_1, k_2 : A \to \texttt{itree}\ \texttt{E}\ R \quad t \cong u}{\texttt{Vis}\ e\ k_1 \cong \texttt{Tau}\ e\ k_2}\ \texttt{EqVis} \tag{4.12}$$

define the α_R function

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

4.1.5 Zip Function

We want the diagram in Figure 4.3 to commute, meaning we get the computation rules

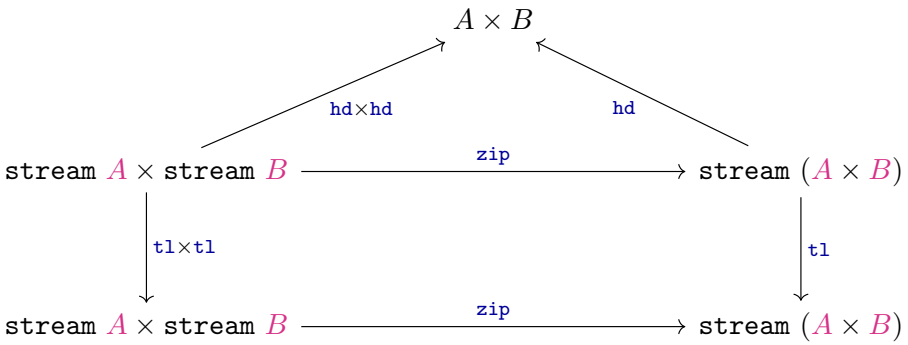

Figure 4.3: TODO

$$(\texttt{hd} \times \texttt{hd}) \equiv \texttt{hd} \circ \texttt{zip} \tag{4.13}$$

$$\texttt{zip} \circ (\texttt{tl} \times \texttt{tl}) \equiv \texttt{tl} \circ \texttt{zip} \tag{4.14}$$

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

$$\begin{aligned}\texttt{lift}_{\texttt{M}} \left(x : \prod_{n:\mathbb{N}} (A \to \texttt{P}_S{}^n\ \texttt{1})\right) \left(u : \prod_{n:\mathbb{N}} (A \to \pi_n(x_{n+1}a) \equiv x_n a)\right) (a : A) : \texttt{M}\ S :=\\ (\lambda\, n, x\ n\ a), (\lambda\, n\ i, p\ n\ a\ i).\end{aligned} \tag{4.15}$$

To use this definition, we first define some helper functions

$$\texttt{zip}_X\ n\ (x, y) = \begin{cases} \texttt{1} & \texttt{if } n = 0 \\ (\texttt{hd}\ x, \texttt{hd}\ y), (\lambda \_, \texttt{zip}_X\ m\ (\texttt{tl}\ x, \texttt{tl}\ y)), & \texttt{if } n = m + 1 \end{cases} \tag{4.16}$$

$$\texttt{zip}_\pi\ n\ (x, y) = \begin{cases} \texttt{refl} & \texttt{if } n = 0 \\ \lambda\, i, (\texttt{hd}\ x, \texttt{hd}\ y), (\lambda \_, \texttt{zip}_\pi\ m\ (\texttt{tl}\ x, \texttt{tl}\ y)\ i), & \texttt{if } n = m + 1 \end{cases}, \tag{4.17}$$

we can then define

$$\texttt{zip}_{lift}\ (x, y) := \texttt{lift}_{\texttt{M}}\ \texttt{zip}_X\ \texttt{zip}\ (x, y). \tag{4.18}$$

Equality of Zip Definitions

We would expect that the two definitions for zip are equal

$$\texttt{transport}_?\ a \equiv \texttt{zip}_{lift}\ a \tag{4.19}$$

$$\equiv \texttt{lift}_{\texttt{M}}\ \texttt{zip}_X\ \texttt{zip}_\pi\ (x, y) \tag{4.20}$$

$$\equiv (\lambda\, n, \texttt{zip}_X\ n\ (x, y)), (\lambda\, n\ i, \texttt{zip}_\pi\ n\ (x, y)\ i) \tag{4.21}$$

zero case X

$$\texttt{zip}_X\ 0\ (x, y) \equiv \mathbf{1} \tag{4.22}$$

Sucessor case X

$$\begin{aligned}
\texttt{zip}_X\ (m+1)\ (x, y) &\equiv (\texttt{hd}\ x, \texttt{hd}\ y), (\lambda _, \texttt{zip}_X\ m\ (\texttt{tl}\ x, \texttt{tl}\ y)) && (4.23)\\
&\equiv (\texttt{hd}\ x, \texttt{hd}\ y), (\lambda _, ?\ (\texttt{tl}\ a)) && (4.24)\\
&\equiv (\texttt{hd}\ (\texttt{transport}_?a)), (\lambda _, \texttt{transport}_?(\texttt{tl}\ a)) && (4.25)\\
&\equiv \texttt{transport}_?\ a && (4.26)\\
& && (4.27)
\end{aligned}$$

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

$$\begin{aligned}
&\equiv (), (\lambda\, i, \texttt{zip}_\pi\ 0\ (x, y)\ i) && (4.28)\\
&\equiv \mathbf{1}, \texttt{refl} && (4.29)\\
& && (4.30)
\end{aligned}$$

successor case

$$\begin{aligned}
&\equiv (\texttt{zip}_X\ (m+1)\ (x, y)), (\lambda\, i, \texttt{zip}_\pi\ (m+1)\ (x, y)\ i) && (4.31)\\
&\equiv ((\texttt{hd}\ x, \texttt{hd}\ y), (\lambda _, \texttt{zip}_X\ m\ (\texttt{tl}\ x, \texttt{tl}\ y))), (\lambda\, i, (\texttt{hd}\ x, \texttt{hd}\ y), (\lambda _, \texttt{zip}_\pi\ m\ (\texttt{tl}\ x, \texttt{tl}\ y)\ i)) && (4.32)
\end{aligned}$$

Complete this proof

4.1.6 Examples of Fixed Points

Zeros

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

$$\texttt{const}_\texttt{X}\ (n : \mathbb{N})\ (c : \mathbb{N}) := \begin{cases} \mathbf{1} & n = 0 \\ (c, \lambda \_, \texttt{const}_\texttt{X}\ m\ c) & n = m+1 \end{cases} \tag{4.33}$$

$$\texttt{const}_\pi\ (n : \mathbb{N})\ (c : \mathbb{N}) := \begin{cases} \texttt{refl} & n = 0 \\ \lambda\, i, (c, \lambda \_, \texttt{const}_\pi\ m\ c\ i) & n = m+1 \end{cases} \tag{4.34}$$

to get the definition of zero stream

$$\texttt{zeros} := \texttt{lift}_\texttt{M}\ \texttt{const}_\texttt{X}\ \texttt{const}_\pi\ 0. \tag{4.35}$$

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

$$\begin{aligned}
\texttt{hd zeros} &\equiv 0 && (4.36)\\
\texttt{tl zeros} &\equiv \texttt{zeros} && (4.37)
\end{aligned}$$

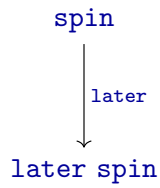

Figure 4.4: TODO

Spin

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 $\mathtt{M}$-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

4.2 Stream Formalization using M-types

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

$$\texttt{stream}\ A := \mathtt{M}_{(A\,,\,\lambda\_,\mathbf{1})} \tag{4.38}$$

When taking the head of a stream, we get

$$\begin{aligned}
\texttt{hd}\ (\texttt{cons}\ x\ xs) &\equiv \pi_1\ \texttt{out}\ (\texttt{cons}\ x\ xs) && (4.39)\\
&\equiv \pi_1\ \texttt{out}\ (\texttt{in}\ (x, \lambda_, xs)) && (4.40)\\
&\equiv \pi_1\ (x, \lambda_, xs) && (4.41)\\
&\equiv x && (4.42)
\end{aligned}$$

and similarly for the tail of the stream

$$\begin{aligned}
\texttt{tl}\ (\texttt{cons}\ x\ xs) &\equiv \pi_2\ \texttt{out}\ (\texttt{cons}\ x\ xs) && (4.43)\\
&\equiv \pi_2\ \texttt{out}\ (\texttt{in}\ (x, \lambda_, xs)) && (4.44)\\
&\equiv \pi_2\ (x, \lambda_, xs) && (4.45)\\
&\equiv xs && (4.46)
\end{aligned}$$

and the other direction is also true

$$\begin{aligned}
\texttt{cons}(\texttt{hd}\ s, \texttt{tl}\ s) &\equiv \texttt{in}\ (\texttt{hd}\ s, \texttt{tl}\ s) && (4.47)\\
&\equiv \texttt{in}\ (\pi_1\ (\texttt{out}\ s), \pi_2\ (\texttt{out}\ s)) && (4.48)\\
&\equiv \texttt{in}\ (\texttt{out}\ s) && (4.49)\\
&\equiv s. && (4.50)
\end{aligned}$$

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

$$\texttt{cons}' f = \texttt{lift}_{\mathtt{M}}\ (\lambda\, c\, n, f\ c) \tag{4.51}$$

$$\texttt{cons}' f = \texttt{lift}_{\mathtt{M}}\ (\lambda\, c\, n, f\ c) \tag{4.52}$$

4.3 ITrees as M-types

We want the following rules for ITrees

$$\frac{r : R}{\texttt{Ret}\ r : \texttt{itree E}\ R}\ \texttt{I}_{\texttt{Ret}} \tag{4.53}$$

$$\frac{A : \mathcal{U} \quad a : \texttt{E}\ A \quad f : A \to \texttt{itree E}\ R}{\texttt{Vis}\ a\ f : \texttt{itree E}\ R}\ \texttt{I}_{\texttt{Vis}}. \tag{4.54}$$

Elimination rules

$$\frac{t : \texttt{itree E}\ R}{\texttt{Tau}\ t : \texttt{itree E}\ R}\ \texttt{E}_{\texttt{Tau}}. \tag{4.55}$$

4.3.1 Delay Monad

check this statement

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

$$\texttt{P}_S(X) = \sum_{x : \mathbf{1}+R} \lambda\{\texttt{inl}\ \_ \to \mathbf{1}; \texttt{inr}\ \_ \to \mathbf{0}\}\ x \to X, \tag{4.56}$$

which is equal to

$$\texttt{P}_S(X) = X + R \times (\mathbf{0} \to X). \tag{4.57}$$

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

$$\texttt{P}_S(X) = X + R \tag{4.58}$$

meaning we get the following diagram.

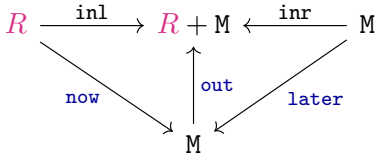

Figure 4.5: Delay monad

(Later = Tau, Ret = Now)

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

4.3.2 Tree

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

$$S = \left(R + \sum_{A : \mathcal{U}} \texttt{E}\ A\ ,\ \lambda\{\texttt{inl}\ \_ \to \mathbf{0}\ ;\ \texttt{inr}\ (A, e) \to A\}\right). \tag{4.59}$$

This will give us the polynomial functor

$$\mathtt{P}_S(X) = \sum_{x:R+\sum_{A:\mathcal{U}} \mathtt{E}\ A} \lambda\{\mathtt{inl}\ \_ \to \mathbf{0}\ ;\ \mathtt{inr}\ (A, e) \to A\}\ x \to X, \tag{4.60}$$

which simplifies to

$$\mathtt{P}_S(X) = (R \times (\mathbf{0} \to X)) + (\sum_{A:\mathcal{U}} \mathtt{E}\ A \times (A \to X)), \tag{4.61}$$

and further

$$\mathtt{P}_S(X) = R + \sum_{A:\mathcal{U}} \mathtt{E}\ A \times (A \to X). \tag{4.62}$$

We get the following diagram for the P-coalgebra.

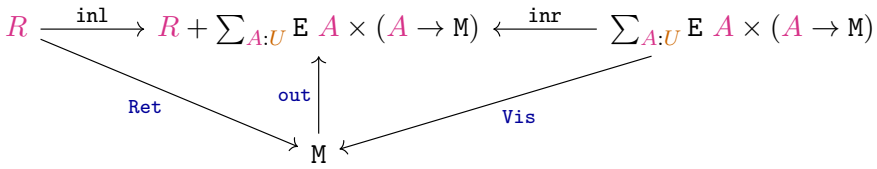

Figure 4.6: TODO

Again we can define `Ret` and `Vis` using the `in` function.

4.3.3 ITrees

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

$$S = \left(\mathbf{1} + R + \sum_{A:\mathcal{U}} (\mathtt{E}\ A)\ \ ,\ \ \lambda\{\mathtt{inl}\ (\mathtt{inl}\ \_) \to \mathbf{1}\ ;\ \mathtt{inl}\ (\mathtt{inr}\ \_) \to \mathbf{0}\ ;\ \mathtt{inr}(A, \_) \to A\}\right). \tag{4.63}$$

Such that the (reduced) polynomial functor becomes

$$\mathtt{P}_S(X) = X + R + \sum_{A:\mathcal{U}} ((\mathtt{E}\ A) \times (A \to X)) \tag{4.64}$$

Giving us the diagram

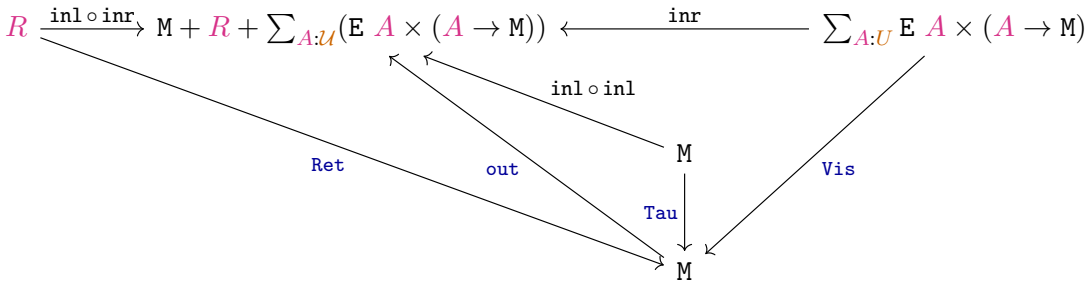

Figure 4.7: TODO

Chapter 5

Additions to the Cubical Agda Library

5.1 Σap

$$\texttt{Σap} : \left(\sum_{x:X} Y x \equiv \sum_{x':X'} Y' x'\right) \equiv \left(\sum_{p:X \equiv X'} Y \equiv_p Y'\right) \tag{5.1}$$

Describe the proof of this? / Is this relevant / Should it be in the appendix?

Chapter 6

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