(Q)M-types and Coinduction in HoTT / CTT

Master's Thesis, Computer Science

Lasse Letager Hansen, 201508114

Supervisor: Bas Spitters

Aarhus University

June 26, 2020

AARHUS
UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

Overview

Goals

- Formalize coinductive types as M-types
- Define equality for coinductive types
- Explore ways to define quotiented M-types

M-types

- M-types are non-wellfounded trees
- M-types are final coalgebras

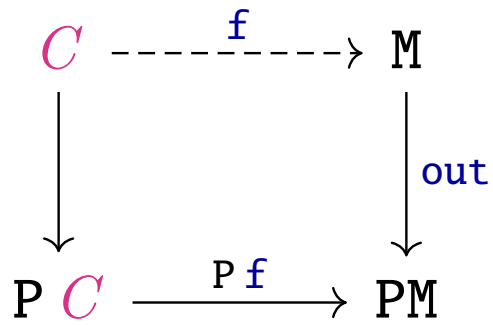
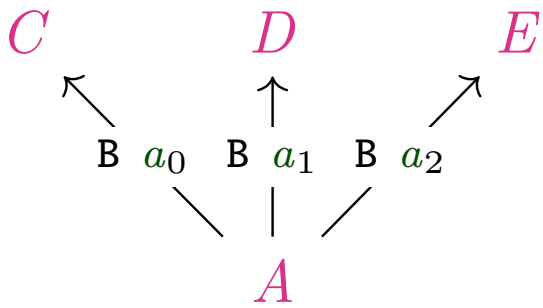


Figure: Final P-Coalgebra

Containers and Polynomial functors

Definition

A Container (or signature) is a dependent pair $S = (A\,, \mathtt{B})$ for the types $A : \mathcal{U}$ and $\mathtt{B} : A \to \mathcal{U}$.



Containers and Polynomial functors

Definition

A polynomial functor $\mathtt{P}_S$ (or extension) for a container $S = (A \,, \mathtt{B})$ is defined, for types as

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

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

$$\mathtt{P}_S\, \mathtt{f}\ (a, \mathtt{g}) = (a, \mathtt{f} \circ \mathtt{g}). \tag{2}$$

Chain

Definition (Chain)

We define a chain as a family of morphisms $\pi_{(n)} : X_{n+1} \to X_n$, over a family of types X_n. See figure.

$$X_0 \xleftarrow[\pi_{(0)}]{} X_1 \xleftarrow[\pi_{(1)}]{} \cdots \xleftarrow[\pi_{(n-1)}]{} X_n \xleftarrow[\pi_{(n)}]{} X_{n+1} \xleftarrow[\pi_{(n+1)}]{} \cdots$$

Definition

The limit of a chain is given as

$$\mathcal{L} = \sum_{(x:\prod_{(n:\mathbb{N})} X_n)} \prod_{(n:\mathbb{N})} (\pi_{(n)}\ x_{n+1} \equiv x_n) \tag{3}$$

Equality between $\mathcal{L}$ and $\mathrm{P}\,\mathcal{L}$

Theorem

Given a container $(A\,, \mathrm{B})$*, we define a chain as the repeated application of* P *to the unit element* $X_n = \mathrm{P}^n\,\mathbf{1}$*, and* $\pi_{(n)} = \mathrm{P}^n\,!$ *where* $! : A \to \mathbf{1}$ *is the unique function into the unit type. Then there is an equality*

$$\mathit{shift} : \mathcal{L} \equiv \mathrm{P}\mathcal{L} \tag{4}$$

where $\mathcal{L}$ *is the limit of this chain.*

Proof structure

The proof is done using the two helper lemmas

$$\alpha : \mathcal{L}^{\mathrm{P}} \equiv \mathrm{P}\,\mathcal{L} \tag{5}$$

$$\mathcal{L}\mathit{unique} : \mathcal{L} \equiv \mathcal{L}^{\mathrm{P}} \tag{6}$$

where $\mathcal{L}^{\mathrm{P}}$ is the limit of the shifted chain defined as ${X'}_n = X_{n+1}$ and ${\pi'}_{(n)} = \pi_{(n+1)}$. With these two lemmas we get $\mathit{shift} = \alpha \cdot \mathcal{L}\mathit{unique}$.

M-types are final

Theorem

The M-type $\mathtt{M}_S$ *is defined as the limit for a polynomial functor* $\mathtt{P}_S$*. This definition fulfills the requirement that* $\mathtt{Final}_S\ \mathcal{L}$*.*

Proof.

By unfolding the definition, we need to show

$$\prod_{(C\text{-}\gamma:\mathtt{Coalg}_S)} \mathtt{isContr}\ (C\text{-}\gamma \Rightarrow \mathcal{L}\text{-}\mathtt{out}) \tag{7}$$

We do this by showing $(C\text{-}\gamma \Rightarrow \mathcal{L}\text{-}\mathtt{out}) \equiv \mathbf{1}$.

Overview

Example: Delay Monad

We define a container

$$(R + \mathbf{1}\,, [\mathbf{0}, \mathbf{1}]) \tag{8}$$

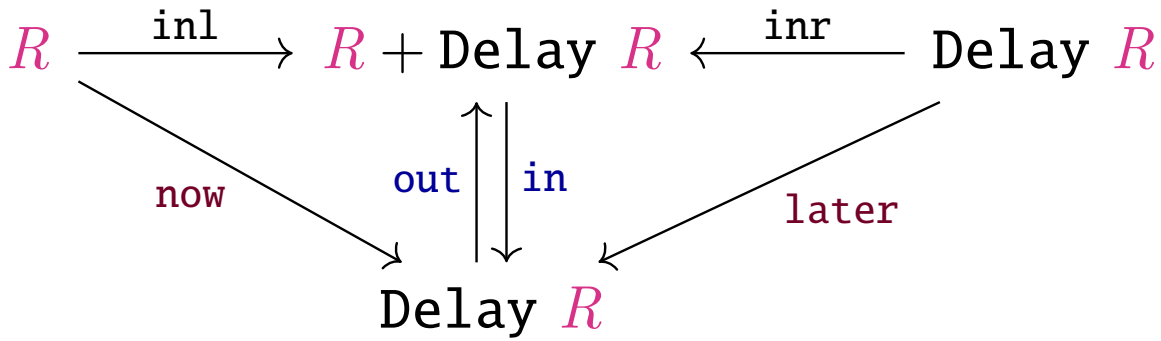
and a polynomial functor

$$\mathtt{P}\, X = \sum_{(x : R + \mathbf{1})} \begin{cases} \mathbf{0} & x = \mathtt{inl}\ r \\ \mathbf{1} & x = \mathtt{inr}\ \star \end{cases} \to X, \tag{9}$$

which simplifies to

$$\mathtt{P}_S\, X = R + X \tag{10}$$

such that we get the diagram



Rules for Constructing M-types

Adding containers $(A\,, \mathtt{B})$ and $(C\,, \mathtt{D})$ for two constructors together is done by

$$\left(A + C\,, \begin{cases} \mathtt{B}\; a & \mathtt{inl}\; a \\ \mathtt{D}\; c & \mathtt{inr}\; c \end{cases} \right) \tag{11}$$

whereas adding containers for two destructors is done by

$$(A \times C\,, \lambda\,(a, c),\ \mathtt{B}\; a + \mathtt{D}\; c) \tag{12}$$

However combining both destructors and constructors is not as simple, which is also the case if we use records or other constructions to define such types.

M-types and Records

We can take a coinductive record and transform it to an M-type by

- Adding all fields that are not a dependent $f_1 : F_1, f_2 : F_2, \ldots$ to the first part of the container as a product
- Adding all fields that are a dependent $d_1 : D_1, d_2 : D_2, \ldots$ to the first part of the container as a dependent product
- Adding all the fields that depend on other fields that depends on previously defined fields $b_1 : \mathtt{B}_1\ d_1, \ldots$
- Converting all recursive fields to the last part of the container $r_1 : R_1 \to \mathtt{M}, \ldots$
- Converting all dependent recursive fields to the last part of the container $z_1 : \prod_{v_1 : V_1} \mathtt{Z}_1\ v_1 \to \mathtt{M}, \ldots$

$$\left(F_1 \times .. \times \sum_{d_1 : D_1} \mathtt{B}_1\ d_1 \times \sum_{d_2 : D_2} .., \lambda \_, R_1 \times .. \times \sum_{v_1 : V_1} \mathtt{Z}_1\ v_1 \times .. \right) \tag{13}$$

Overview

Example: Streams

We can now define streams for a given type A. We start with a container

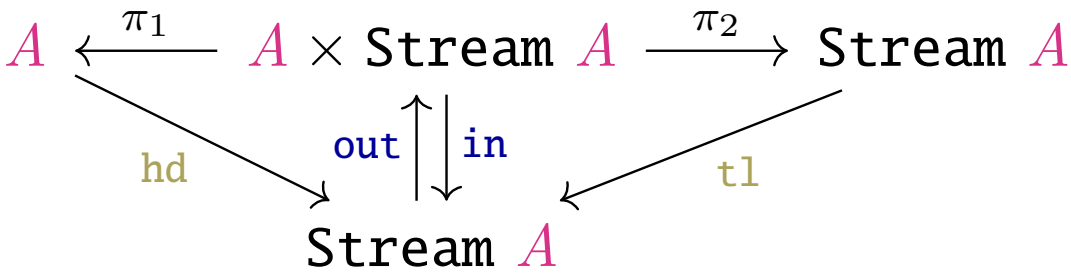
$$(A \,, \mathbf{1}) \tag{14}$$

For which we get the polynomial functor

$$\mathtt{P}\, X = A \times X \tag{15}$$

For which the we get the $\mathtt{M}$-type for stream

$$\mathtt{Stream}\ A = A \times \mathtt{Stream}\ A \tag{16}$$



Bisimulation for Streams

We can define an equivalence relation

$$\frac{\texttt{hd}\ s \equiv \texttt{hd}\ t \quad \texttt{tl}\ s \sim_{\texttt{stream}} \texttt{tl}\ t}{s \sim_{\texttt{stream}} t} \tag{17}$$

We can convert this to an equality

Bisimulation

Definition

A relation $\mathcal{R} : C \to C \to \mathcal{U}$ for a coalgebra $C\text{-}\gamma : \mathtt{Coalg}_S$, is a (strong) bisimulation relation if the type $\overline{\mathcal{R}} = \sum_{(a:C)} \sum_{(b:C)} a\ \mathcal{R}\ b$ and the function $\alpha_{\mathcal{R}} : \overline{\mathcal{R}} \to \mathtt{P}_S\, \overline{\mathcal{R}}$ forms a $\mathtt{P}_S$-coalgebra $\overline{\mathcal{R}}\text{-}\alpha_{\mathcal{R}} : \mathtt{Coalg}_S$, making the diagram bellow commute ($\Longrightarrow$ represents $\mathtt{P}_S$-coalgebra morphisms). That is

$$\gamma \circ \pi_1^{\overline{\mathcal{R}}} \equiv \mathtt{P}_S\, \pi_1^{\overline{\mathcal{R}}} \circ \alpha_{\mathrm{R}} \tag{18}$$

and similarly for $\pi_2^{\overline{\mathcal{R}}}$.

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

Coinduction Principle

Theorem (Coinduction principle)

Given a relation $\mathcal{R}$, that is a bisimulation for a M*-type, then (strongly) bisimilar elements $x\ \mathcal{R}\ y$ are equal $x \equiv y$.*

Proof.

Given a relation $\mathcal{R}$ that is bisimulation relation over a final P-coalgebra $\mathtt{M\text{-}out} : \mathtt{Coalg}_S$ we get the diagram

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

By the finality of M-out, we get a function $!$ from M to $\overline{R}$, which is unique, meaning $\pi_1^{\overline{\mathcal{R}}} \equiv\ ! \equiv \pi_2^{\overline{\mathcal{R}}}$. Now given $r : x\ \mathcal{R}\ y$, we can construct the equality

$$x \equiv \pi_1^{\overline{\mathcal{R}}}(x, y, r) \equiv \pi_2^{\overline{\mathcal{R}}}(x, y, r) \equiv y, \tag{19}$$

giving us the coinduction principle for M-types. □

Overview

Propositional Truncation and Set Truncated Quotient

A Higher Inductive Type (HIT) is a type defined by point constructors as well as equality constructors. We define propositional truncation by

Definition (Propositional Truncation)

$$\frac{x : A}{|\,x\,| : \|A\|} \quad (20) \qquad \frac{x, y : \|A\|}{\mathtt{squash}\ x\ y : x \equiv y} \quad (21)$$

We can define set truncated quotients as the following HIT.

Definition (Set Truncated Quotient)

$$\frac{x : A}{[\,x\,] : A/\mathcal{R}} \quad (22) \qquad \frac{x, y : A/\mathcal{R} \quad r : x\ \mathcal{R}\ y}{\mathtt{eq}/\ x\ y\ r : x \equiv y} \quad (23)$$

$$\frac{}{\mathtt{squash}/ : \mathtt{isSet}\ (A/\mathrm{R})} \quad (24)$$

Quotient Inductive-Inductive Type (QIIT)

A QIIT is a type that is defined at the same time as a relation. That is

- Constructors may refer to previously defined constructors (recursive).
- Constructors can be a point constructor or equality constructor.
- Constructors may refer to constructors of the relation.
- The type is set truncated.

And similarly for the constructors of the relation.

Partiality monad

QM-type

We would like to model equality of programs as two programs being equal if one terminates with a value then the other also terminates with the same value a finite number of steps later. This is model by quotienting the delay monad by a relation defined by the constructors

$$\frac{x \sim y}{\texttt{later}\ x \sim y}\ \sim_{\texttt{later}_\texttt{l}} \quad (25) \qquad \frac{x \sim y}{x \sim \texttt{later}\ y}\ \sim_{\texttt{later}_\texttt{r}} \quad (26)$$

$$\frac{a \equiv b}{\texttt{now}\ a \sim \texttt{now}\ b}\ \sim_{\texttt{now}} \quad (27) \qquad \frac{x \sim y}{\texttt{later}\ x \sim \texttt{later}\ y}\ \sim_{\texttt{later}} \quad (28)$$

This gives us a construction for a QM-types.

$$\texttt{Delay}\ R/\sim \quad (29)$$

Partiality Monad

QIIT

We can define it by type constructors

$$\frac{}{R_\perp : \mathcal{U}} \quad (30) \qquad \frac{}{\perp : R_\perp} \quad (31) \qquad \frac{a : R}{\eta\ a : R_\perp} \quad (32)$$

and an ordering relation $(\cdot \sqsubseteq_\perp \cdot)$ indexed twice over $R_\perp$

$$\frac{x : R_\perp}{x \sqsubseteq_\perp x}\ \sqsubseteq\texttt{refl} \quad (33) \qquad \frac{x \sqsubseteq_\perp y \quad y \sqsubseteq_\perp z}{x \sqsubseteq_\perp z}\ \sqsubseteq\texttt{trans} \quad (34)$$

$$\frac{x, y : R_\perp \quad p : x \sqsubseteq_\perp y \quad q : y \sqsubseteq_\perp x}{\alpha_\perp\ p\ q : x \equiv y} \quad (35)$$

where $\perp$ is lowest in the order

$$\frac{x : R_\perp}{\perp \sqsubseteq_\perp x}\ \sqsubseteq\texttt{never} \quad (36)$$

with an upper bound

$$\frac{\mathtt{s} : \mathbb{N} \to R_\perp \quad \mathtt{b} : \prod_{(n:\mathbb{N})} \mathtt{s}_n \sqsubseteq_\perp \mathtt{s}_{n+1}}{\bigsqcup (\mathtt{s}, \mathtt{b}) : R_\perp} \tag{37}$$

which gives a bound for a sequence

$$\frac{\mathtt{s} : \mathbb{N} \to R_\perp \quad \mathtt{b} : \prod_{(n:\mathbb{N})} \mathtt{s}_n \sqsubseteq_\perp \mathtt{s}_{n+1}}{\prod_{(n:\mathbb{N})} \mathtt{s}_n \sqsubseteq_\perp \bigsqcup(\mathtt{s}, \mathtt{b})} \tag{38}$$

and which is a least upper bound

$$\frac{\prod_{(n:\mathbb{N})} \mathtt{s}_n \sqsubseteq_\perp x}{\bigsqcup(\mathtt{s}, \mathtt{b}) \sqsubseteq_\perp x} \tag{39}$$

and finally set truncated by the constructor $(-)_\perp$-isSet

Partiality Monad

Equality

To show these two definitions are equal we

- Define an intermediate type of sequences

$$\mathsf{Seq}\ A = \sum_{(\mathsf{s}:\mathbb{N}\to A+\mathbf{1})} \prod_{(n:\mathbb{N})} \mathsf{s}_n \sqsubseteq_{\mathsf{R}+\mathbf{1}} \mathsf{s}_{n+1} \tag{40}$$

- Show that the Delay monad is equal to this intermediate type
- Define weak bisimilarty for sequences, and show it respects the equality to the Delay monad
- Show that $\mathsf{Seq}_R/\sim$ is equal to the partiality monad, by
  - Defining a function from $\mathsf{Seq}_R/\sim$ to $R_\perp$
  - Show this function is injective and surjective

However the proof for surjectivity requires the axiom of countable choice!

We can see that the QIIT for the partiality monad is not trivial, however we can define quotients as QIITs rather trivially, however this would *only* give us the constructors

$$\frac{}{\texttt{now}\ a} \tag{41}$$

$$\frac{x}{\texttt{later}\ x} \tag{42}$$

$$\frac{x \equiv y}{\texttt{later}\ x \equiv y}\ \texttt{later}_{\equiv} \tag{43}$$

What do we want from a quotient M-type?

- We would like to be able to construct a quotient from an M-type and a relation.
- We should be able to take an element to its equality class without the axiom of choice
- It should be equal to the set truncated quotient, if we assume the axiom of choice?

Alternative: Quotient Polynomial Functor (QPF)

We can define quotiented $\mathtt{M}$-types from a quotient polynomial functor.

Definition (Quotient Polynomial Functor)

We define a quotient polynomial functor (QPF), for types as

$$\mathtt{F}\ X = \sum_{(a:A)} ((\mathtt{B}\ a \to X)/\sim_a) \tag{44}$$

and for a function $\mathtt{f} : X \to Y$, we use the quotient eliminator with

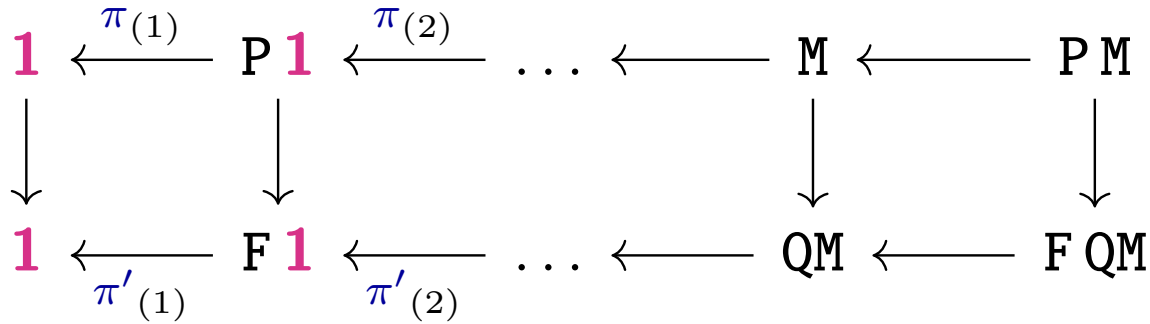
$$\mathtt{P} = \lambda \_, (\mathtt{B}\ a \to Y)/\sim_a \tag{45}$$

for which we need $\sim_{\mathtt{ap}}$, which says that given a function $\mathtt{f}$ and $\mathtt{x} \sim_a \mathtt{y}$ then $\mathtt{f} \circ \mathtt{x} \sim_a \mathtt{f} \circ \mathtt{y}$.

$$\mathtt{F}\ \mathtt{f}\ (a, \mathtt{g}) = (a, \mathtt{elim}\ \mathtt{g}) \tag{46}$$

Alternative: Quotient Polynomial Functor (QPF)

We get the diagram



For which $\mathtt{M} \equiv \mathtt{P}\,\mathtt{M}$ and we would hope $\mathtt{QM} \equiv \mathtt{F}\,\mathtt{QM}$, however this requires the axiom of choice.

Conclusion

We have

- given a formalization/semantic of M-types
- shown examples of and rules for how to construct M-types
- given a coinduction principle for M-types
- described the construction of the partiality monad as a QIIT
- discussed ways of constructing quotient M-types

Contribution

- Formalization in Cubical Agda
- Introducing QM-types

Future Work

- Indexed M-types
- Showing finality of QM-types and fully formalizing the constructions
-