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

Master's Thesis, Computer Science

Lasse Letager Hansen, 201508114

Supervisor: Bas Spitters

Aarhus University

June 29, 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

Containers and Polynomial functors

Definition

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

$$\begin{tikzcd}
C & D & E \\
& A \arrow[ul, "\mathtt{B}\ a_0"] \arrow[u, "\mathtt{B}\ a_1"] \arrow[ur, "\mathtt{B}\ a_2"] &
\end{tikzcd}$$

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

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

We let $\mathtt{M}_S$ be the limit, for a chain defined by $X_n = \mathtt{P}^n\,\mathbf{1}$, and $\pi_{(n)} = \mathtt{P}^n\,!$

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

Theorem

There is an equality

$$shift : \mathtt{M} \equiv \mathtt{P}\,\mathtt{M} \tag{4}$$

from which we can define helper functions

$$\mathtt{in} : \mathtt{P}\,\mathtt{M} \to \mathtt{M} \qquad \mathtt{out} : \mathtt{M} \to \mathtt{P}\,\mathtt{M} \tag{5}$$

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

Theorem

There is an equality

$$shift : \mathtt{M} \equiv \mathtt{P}\,\mathtt{M} \tag{4}$$

from which we can define helper functions

$$\mathtt{in} : \mathtt{P}\,\mathtt{M} \to \mathtt{M} \qquad \mathtt{out} : \mathtt{M} \to \mathtt{P}\,\mathtt{M} \tag{5}$$

Proof structure

The proof is done using the two helper lemmas

$$\alpha : \mathcal{L}^{\mathrm{P}} \equiv \mathrm{P}\,\mathcal{L} \tag{6}$$

$$\mathcal{L}unique : \mathcal{L} \equiv \mathcal{L}^{\mathrm{P}} \tag{7}$$

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 $shift = \alpha \cdot \mathcal{L}unique$.

We want to show that M-types are final coalgebras.

Definition

A P-coalgebra is defined as

$$\sum_{(C:\mathcal{U})} C \to \mathtt{P}\ C \tag{8}$$

which we denote C-γ. We define P-coalgebra morphisms as

$$C\text{-}\gamma \Rightarrow D\text{-}\delta = \sum_{(\mathtt{f}:C\to D)} \delta \circ \mathtt{f} \equiv \mathtt{P}\mathtt{f} \circ \gamma \tag{9}$$

A coalgebra is final, if the following is true

$$\sum_{(D\text{-}\rho)} \prod_{(C\text{-}\gamma)} \mathtt{isContr}\ (C\text{-}\gamma \Rightarrow D\text{-}\rho) \tag{10}$$

$\texttt{M}$-types are final coalgebras

Theorem

$\texttt{M}$-types are final coalgebras. That is $\texttt{Final}_S\ \texttt{M}$.

Theorem

M-types are final coalgebras. That is $\mathtt{Final}_S\ \mathtt{M}$.

Proof structure

The definition of finality is

$$\prod_{(C\text{-}\gamma : \mathtt{Coalg}_S)} \mathtt{isContr}\ (C\text{-}\gamma \Rightarrow \mathtt{M\text{-}out}) \tag{11}$$

which we show by $(C\text{-}\gamma \Rightarrow \mathtt{M\text{-}out}) \equiv \mathbf{1}$.

Overview

Example: Delay Monad

A delay monad is defined by the two constructors

$$\frac{r : R}{\mathtt{now}\ r : \mathtt{Delay}\ R} \quad (12) \qquad\qquad \frac{t : \mathtt{Delay}\ R}{\mathtt{later}\ t : \mathtt{Delay}\ R} \quad (13)$$

We define a container

$$(R + \mathbf{1}\,, [\mathbf{0}, \mathbf{1}]) \tag{14}$$

and a polynomial functor

$$\mathtt{P}\ X = \sum_{(x : R + \mathbf{1})} \begin{cases} \mathbf{0} & x = \mathtt{inl}\ r \\ \mathbf{1} & x = \mathtt{inr}\ \star \end{cases} \to X \quad = R + X, \tag{15}$$

such that we get the diagram

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

Rules for Constructing $\mathtt{M}$-types

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

$$\left(A + C\,,\ \begin{cases} \mathtt{B}\ a & \mathtt{inl}\ a \\ \mathtt{D}\ c & \mathtt{inr}\ c \end{cases}\right) \tag{16}$$

whereas adding containers for two destructors is done by

$$(A \times C\,, \lambda\,(a, c),\ \mathtt{B}\ a + \mathtt{D}\ c) \tag{17}$$

However combining both destructors and constructors is not as simple. Which is similar to rules for coinductive records.

M-types and Records

We can take a coinductive record and transform it to an M-type. The types of fields in a coinductive records are

- non-dependent fields
- dependent fields
- recursive fields
- dependent and recursive fields

We will give some examples of these.

Lets try and convert the record

$$\begin{aligned}&\texttt{record}\ \mathit{tree} : \mathcal{U}\ \texttt{where}\\&\qquad \mathit{value} : \mathbb{N}\\&\qquad \mathit{left\text{-}child} : \mathit{tree}\\&\qquad \mathit{right\text{-}child} : \mathit{tree}\end{aligned} \tag{18}$$

To an $\mathtt{M}$-type defined by the container

$$(\mathbb{N}\,, \mathbf{1} + \mathbf{1}) \tag{19}$$

Lets try and convert the record

$$\begin{aligned} &\texttt{record}\ bet : \mathcal{U}\ \texttt{where} \\ &\qquad value_a : \mathbb{N} \\ &\qquad value_b : \mathbb{N} \\ &\qquad winner : value_a \leq value_b \to bool \end{aligned} \tag{20}$$

To an M-type defined by the container

$$\left(\sum_{(value_a : \mathbb{N})} \sum_{(value_b : \mathbb{N})} value_a \leq value_b \to bool \, , \mathbf{0} \right) \tag{21}$$

Lets try and convert the record

$$\begin{aligned} &\texttt{record example } A : \mathcal{U} \texttt{ where} \\ &\qquad \mathit{value} : A \\ &\qquad \mathit{index\text{-}type} : \mathcal{U} \\ &\qquad \mathit{continue} : \mathit{index\text{-}type} \to \texttt{example } A \end{aligned} \tag{22}$$

To an M-type defined by the container

$$(A \times \mathcal{U}\, , \lambda\, (\_, \mathit{index\text{-}type}), \mathit{index\text{-}type}) \tag{23}$$

Overview

Example: Streams

We can now define streams for a given type A, as a records

$$\begin{aligned} &\texttt{record Stream } A : \mathcal{U} \texttt{ where} \\ &\quad hd : A \\ &\quad tl : \texttt{Stream } A \end{aligned} \tag{24}$$

corresponding to the container

$$(A\,, \mathbf{1}) \tag{25}$$

for which we get the polynomial functor

$$\texttt{P}\; X = A \times X \tag{26}$$

For which the we get the M-type for stream

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

Bisimulation for Streams

We can define an equivalence relation

$$\frac{\texttt{hd}\ s \equiv \texttt{hd}\ t \quad \texttt{tl}\ s \sim_{\texttt{stream}} \texttt{tl}\ t}{s \sim_{\texttt{stream}} t} \tag{27}$$

We can formalize this as a bisimulation. A (strong) bisimulation for a P-coalgebra C-γ is given by

- a relation $\mathcal{R} : C \to C \to \mathcal{U}$
- a type $\overline{\mathcal{R}} = \sum_{(a:C)} \sum_{(b:C)} a\ \mathcal{R}\ b$
- and a function $\alpha_{\mathcal{R}} : \overline{\mathcal{R}} \to \mathtt{P}_S\ \overline{\mathcal{R}}$

Such that $\overline{\mathcal{R}}$-$\alpha_{\mathcal{R}}$ is a P-coalg, making the following diagram commute.

$$C\text{-}\gamma \xLeftarrow{\pi_1{}^{\overline{\mathcal{R}}}} \overline{\mathcal{R}}\text{-}\alpha_{\mathcal{R}} \xRightarrow{\pi_2{}^{\overline{\mathcal{R}}}} C\text{-}\gamma$$

In MLTT this is not enough to define an equality, however in HoTT and CTT it is.

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* $x \equiv y$.

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.

We get the diagram

$$\texttt{M-out} \xLeftarrow{\pi_1{}^{\overline{\mathcal{R}}}} \overline{\mathcal{R}}\text{-}\alpha_{\mathcal{R}} \xRightarrow{\pi_2{}^{\overline{\mathcal{R}}}} \texttt{M-out}$$

Since `M-out` is a final coalgebra, functions into it are unique, meaning

$$\pi_1^{\overline{\mathcal{R}}} \equiv \pi_2^{\overline{\mathcal{R}}} \tag{28}$$

therefore given $r : x \; \mathcal{R} \; y$, we can construct the equality

$$x \equiv \pi_1^{\overline{\mathcal{R}}}(x, y, r) \equiv \pi_2^{\overline{\mathcal{R}}}(x, y, r) \equiv y. \tag{29}$$

□

Overview

Set Truncated Quotient

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

Set Truncated Quotient

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

Set Truncated Quotient

A Higher Inductive Type (HIT) is a type defined by point constructors as well as equality constructors. We can define set truncated quotients as the following HIT.

Definition (Set Truncated Quotient)

$$\frac{x : A}{[\, x \,] : A / \mathcal{R}} \quad (30) \qquad \frac{x, y : A \quad r : x \;\mathcal{R}\; y}{\mathbf{eq/}\; x\; y\; r : [\, x \,] \equiv [\, y \,]} \quad (31)$$

$$\frac{}{\texttt{squash/} : \texttt{isSet}\ (A / \texttt{R})} \quad (32)$$

Partiality monad

The partiality monad describes equality of partial computations.

Partiality monad

QM-type

The partiality monad describes equality of partial computations.

We define the partiality monad by quotienting the delay monad by a relation defined by the constructors

$$\frac{x \sim y}{\texttt{later}\ x \sim y} \sim_{\texttt{later}_\texttt{l}} \quad (33) \qquad \frac{x \sim y}{x \sim \texttt{later}\ y} \sim_{\texttt{later}_\texttt{r}} \quad (34)$$

$$\frac{a \ \equiv \ b}{\texttt{now}\ a \sim \texttt{now}\ b} \sim_{\texttt{now}} \quad (35) \qquad \frac{x \sim y}{\texttt{later}\ x \sim \texttt{later}\ y} \sim_{\texttt{later}} \quad (36)$$

the partiality monad is then given by the set truncated quotient

$$\texttt{Delay}\ R/\sim \tag{37}$$

however we need the axiom of (countable) choice

Partiality Monad
QIIT

A QIIT is a type that is defined at the same time as a relation and then set truncated.

A QIIT is a type that is defined at the same time as a relation and then set truncated.

We can define the partiality monad as a QIIT with type constructors

$$\frac{}{R_\perp : \mathcal{U}} \quad (38) \qquad \frac{}{\perp : R_\perp} \quad (39) \qquad \frac{a : R}{\eta\ a : R_\perp} \quad (40)$$

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

$$\frac{x : R_\perp}{x \sqsubseteq_\perp x} \sqsubseteq_{\texttt{refl}} \quad (41) \qquad \frac{x \sqsubseteq_\perp y \quad y \sqsubseteq_\perp z}{x \sqsubseteq_\perp z} \sqsubseteq_{\texttt{trans}} \quad (42)$$

$$\frac{x, y : R_\perp \quad p : x \sqsubseteq_\perp y \quad q : y \sqsubseteq_\perp x}{\alpha_\perp\ p\ q : x \equiv y} \tag{43}$$

where $\perp$ is the smallest element in the order

$$\frac{x : R_\perp}{\perp \sqsubseteq_\perp x} \sqsubseteq_{\texttt{never}} \tag{44}$$

with an upper bound

$$\frac{\mathtt{s} : \mathbb{N} \to R_\bot \quad \mathtt{b} : \prod_{(n:\mathbb{N})} \mathtt{s}_n \sqsubseteq_\bot \mathtt{s}_{n+1}}{\bigsqcup(\mathtt{s}, \mathtt{b}) : R_\bot} \tag{45}$$

which gives a bound for a sequence

$$\frac{\mathtt{s} : \mathbb{N} \to R_\bot \quad \mathtt{b} : \prod_{(n:\mathbb{N})} \mathtt{s}_n \sqsubseteq_\bot \mathtt{s}_{n+1}}{\prod_{(n:\mathbb{N})} \mathtt{s}_n \sqsubseteq_\bot \bigsqcup(\mathtt{s}, \mathtt{b})} \tag{46}$$

and which is a least upper bound

$$\frac{\prod_{(n:\mathbb{N})} \mathtt{s}_n \sqsubseteq_\bot x}{\bigsqcup(\mathtt{s}, \mathtt{b}) \sqsubseteq_\bot x} \tag{47}$$

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

Partiality Monad

Equality

To show these two definitions are equal we

- Define an intermediate type of sequences

$$\mathsf{Seq}\ A = \sum_{(\mathsf{s}:\mathbb{N}\to A+\mathbf{1})} \prod_{(n:\mathbb{N})} \mathsf{s}_n \sqsubseteq_{\mathtt{R}+\mathbf{1}} \mathsf{s}_{n+1} \tag{48}$$

- 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!

Simple Partialty Monad QIIT

This construction is quite involved, however doing a trivial construction

$$\frac{a : R}{\texttt{now}\ a : R_\perp} \quad (49) \qquad\qquad \frac{t : R_\perp}{\texttt{later}\ t : R_\perp} \quad (50)$$

$$\frac{x \equiv y}{\texttt{later}\ x \equiv y}\ \texttt{later}_{\equiv} \quad (51)$$

would not give us the intended functionality, so we need to think more about how we define the QIIT.

We have to think about how we define a QM-type.

What do we want from a quotient $\mathtt{M}$-type?

- We would like to be able to construct a quotient from an $\mathtt{M}$-type and a relation.
- We should be able to lift constructors to the quotient without the axiom of choice.
- The type should be equal to the type defined by the set truncated quotient if we assume the axiom of choice.

Alternative: Quotient Polynomial Functor (QPF)

We can define quotiented M-types from a quotient polynomial functor.

Definition (Quotient Polynomial Functor)

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

$$\mathtt{F}\, X = \sum_{(a : A)} ((\mathtt{B}\, a \to X)/\sim_a) \tag{52}$$

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

$$\mathtt{P} = \lambda\, \_, (\mathtt{B}\, a \to Y)/\sim_a \tag{53}$$

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

$$\mathtt{F}\, \mathtt{f}\, (a, \mathtt{g}) = (a, \mathtt{elim}\ \mathtt{g}) \tag{54}$$

Alternative: Quotient Polynomial Functor (QPF)

We get the diagram

$$\begin{tikzcd} \mathbf{1} \arrow[d] & \mathtt{P}\,\mathbf{1} \arrow[l, "\pi_{(1)}"'] \arrow[d] & \ldots \arrow[l, "\pi_{(2)}"'] & \mathtt{M} \arrow[l] \arrow[d] & \mathtt{P}\,\mathtt{M} \arrow[l] \arrow[d] \\ \mathbf{1} & \mathtt{F}\,\mathbf{1} \arrow[l, "\pi'_{(1)}"] & \ldots \arrow[l, "\pi'_{(2)}"] & \mathtt{QM} \arrow[l] & \mathtt{F}\,\mathtt{QM} \arrow[l] \end{tikzcd}$$

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.

Alternative: Quotient Polynomial Functor (QPF)

We get the diagram

$$\begin{tikzcd} \mathbf{1} \arrow[d] & \mathtt{P}\,\mathbf{1} \arrow[l, "\pi_{(1)}"'] \arrow[d] & \ldots \arrow[l, "\pi_{(2)}"'] & \mathtt{M} \arrow[l] \arrow[d] & \mathtt{P}\,\mathtt{M} \arrow[l] \arrow[d] \\ \mathbf{1} & \mathtt{F}\,\mathbf{1} \arrow[l, "\pi'_{(1)}"] & \ldots \arrow[l, "\pi'_{(2)}"] & \mathtt{QM} \arrow[l] & \mathtt{F}\,\mathtt{QM} \arrow[l] \end{tikzcd}$$

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.

However looking further into this is future research.

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 Q`M`-types and fully formalizing the constructions
- Equality between Coinductive records and `M`-types
- Explore Guarded Cubical Type Theory