
M-types and Coinduction in HoTT and Cubical Type Theory

Lasse Letager Hansen, 201912345

Master's Thesis, Computer Science

March 17, 2020

Advisor: Bas Spitters

Abstract

in English...

Resumé

in Danish...

Acknowledgments



*Lasse Letager Hansen,
Aarhus, March 17, 2020.*

Contents

Abstract	iii
Resumé	v
Acknowledgments	vii
1 Introduction	1
2 M-types	3
2.1 Containers / Signatures	3
2.2 ITrees as M-types	4
2.2.1 Delay Monad	5
2.2.2 Tree	5
2.2.3 ITrees	6
2.3 Co-induction Principle for M-types	6
2.3.1 Bisimulation of Streams	7
2.3.2 Bisimulation of Delay monad	7
2.3.3 Bisimulation of ITrees	7
2.4 Quotient M-type	8
2.5 Closure properties of M-types	8
2.5.1 Closure under products	10
2.5.2 Examples of fixed points	11
3 Additions to the Cubical Agda Library	13
3.1 Lemma 10	14
3.2 Lemma 13	15
3.3 Missing postulates	15
4 Conclusion	17
Bibliography	19
A The Technical Details	21

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

$$\begin{aligned} \mathbf{P}_S &: \mathcal{U} \rightarrow \mathcal{U} \\ \mathbf{P}(X) &:= \mathbf{P}_S(X) = \sum_{a:A} B(a) \rightarrow X \end{aligned} \quad (2.1)$$

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

$$\begin{aligned} \mathbf{P}f &: \mathbf{P}X \rightarrow \mathbf{P}Y \\ \mathbf{P}f(a, g) &= (a, f \circ g) \end{aligned} \quad (2.2)$$

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

$$\mathbf{P}_S(X) = \sum_{a:A} \mathbf{1} \rightarrow X \quad (2.3)$$

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

$$\mathbf{P}_S(X) = A \times X \quad (2.4)$$

Now we define the coalgebra for this functor with type

$$\mathbf{Coalg}_S = \sum_{C:\mathcal{U}} C \rightarrow \mathbf{P}C \quad (2.5)$$

we denote a coalgebra of C and γ as $C-\gamma$. The coalgebra morphisms are defined as

$$\begin{aligned} _ \Rightarrow _ &: \mathbf{Coalg}_S \rightarrow \mathbf{Coalg}_S \\ C-\gamma \Rightarrow D-\delta &= \sum_{f:C \rightarrow D} \delta \circ f = \mathbf{P}f \circ \gamma \end{aligned} \quad (2.6)$$

M-types can now be defined from a container \mathcal{S} as the type $\mathbf{M}_{\mathcal{S}}$ such that the coalgebra for $\mathbf{M}_{\mathcal{S}}$ and $\text{out} : \mathbf{M}_{\mathcal{S}} \rightarrow \mathbf{P}_{\mathcal{S}}(\mathbf{M}_{\mathcal{S}})$ fulfills the property

$$\text{Final}_{\mathcal{S}} := \sum_{(X-\rho:\text{Coalg}_{\mathcal{S}})} \prod_{(C-\gamma:\text{Coalg}_{\mathcal{S}})} \text{isContr}(C-\gamma \Rightarrow X-\rho) \quad (2.7)$$

that is $\prod_{(C-\gamma:\text{Coalg}_{\mathcal{S}})} \text{isContr}(C-\gamma \Rightarrow \mathbf{M}_{\mathcal{S}}-\text{out})$. We denote this construction of the M-type as $\mathbf{M}_{(A,B)}$ or $\mathbf{M}_{\mathcal{S}}$ or just \mathbf{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 $\mathbf{P}_{\mathcal{S}}(\mathbf{M}) = A \times \mathbf{M}$, meaning we have the following diagram, where out is an isomorphism (because of the finality of the

$$\begin{array}{ccccc} A & \xleftarrow{\pi_1} & A \times \mathbf{M}_{(A,\lambda_,1)} & \xrightarrow{\pi_2} & \mathbf{M}_{(A,\lambda_,1)} \\ & \searrow \text{hd} & \uparrow \text{out} & \nearrow \text{tl} & \\ & & \mathbf{M}_{(A,\lambda_,1)} & & \end{array}$$

Figure 2.1: M-types of streams

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

$$\frac{A:\mathcal{U} \quad s:\text{stream } A}{\text{hd } s:A} \text{E}_{\text{hd}} \quad (2.8)$$

$$\frac{A:\mathcal{U} \quad s:\text{stream } A}{\text{tl } s:\text{stream } A} \text{E}_{\text{tl}} \quad (2.9)$$

$$\frac{A:\mathcal{U} \quad x:A \quad xs:\text{stream } A}{\text{cons } x \ xs:\text{stream } A} \text{I}_{\text{cons}} \quad (2.10)$$

2.2 ITrees as M-types

We want the following rules for ITrees

$$\frac{r:R}{\text{Ret } r:\text{itree } E \ R} \text{I}_{\text{Ret}} \quad (2.11)$$

$$\frac{A:\mathcal{U} \quad a:E \ A \quad f:A \rightarrow \text{itree } E \ R}{\text{Vis } a \ f:\text{itree } E \ R} \text{I}_{\text{vis}}. \quad (2.12)$$

Elimination rules

$$\frac{t:\text{itree } E \ R}{\text{Tau } t:\text{itree } E \ R} \text{E}_{\text{Tau}}. \quad (2.13)$$

2.2.1 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 $S = (1 + R, \lambda\{\text{inl } _ \rightarrow 1 ; \text{inr } _ \rightarrow 0\})$, we then get the polynomial functor

$$\mathbf{P}_S(X) = \sum_{x:1+R} \lambda\{\text{inl } _ \rightarrow 1 ; \text{inr } _ \rightarrow 0\} x \rightarrow X, \quad (2.14)$$

which is equal to

$$\mathbf{P}_S(X) = X + R \times (0 \rightarrow X). \quad (2.15)$$

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

$$\mathbf{P}_S(X) = X + R \quad (2.16)$$

meaning we get the following diagram.

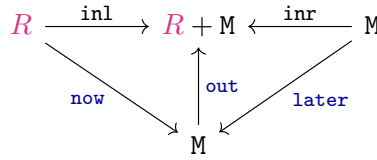


Figure 2.2: Delay monad

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

(Later
= Tau,
Ret =
Now)

2.2.2 Tree

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

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

This will give us the polynomial functor

$$\mathbf{P}_S(X) = \sum_{x:R+\sum_{A:\mathcal{U}} \mathbf{E} A} \lambda\{\text{inl } _ \rightarrow 0 ; \text{inr } (A, e) \rightarrow A\} x \rightarrow X, \quad (2.18)$$

which simplifies to

$$\mathbf{P}_S(X) = (R \times (0 \rightarrow X)) + \left(\sum_{A:\mathcal{U}} \mathbf{E} A \times (A \rightarrow X) \right), \quad (2.19)$$

and further

$$\mathbf{P}_S(X) = R + \sum_{A:\mathcal{U}} \mathbf{E} A \times (A \rightarrow X). \quad (2.20)$$

We get the following diagram for the **P**-coalgebra.

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

check
this
state-
ment

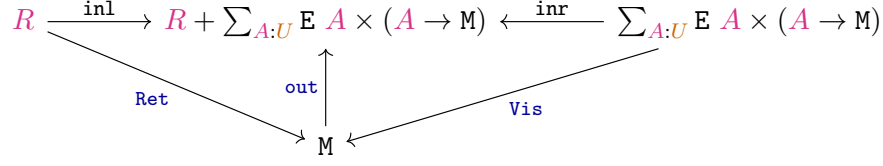


Figure 2.3: TODO

2.2.3 ITrees

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

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

Such that the (reduced) polynomial functor becomes

$$\mathbf{P}_S(X) = X + R + \sum_{A:\mathcal{U}} ((E A) \times (A \rightarrow X)) \quad (2.22)$$

Giving us the diagram

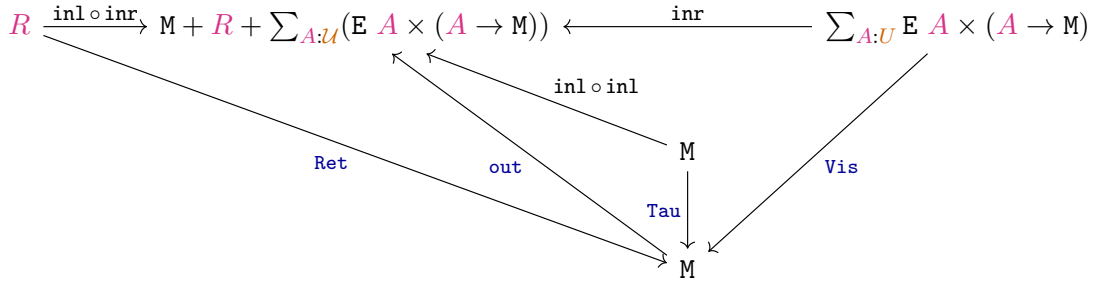


Figure 2.4: TODO

2.3 Co-induction Principle for M-types

We can now construct a bisimulation: forall coalgebras $C-\gamma : \text{Coalg}_S$, if we have a relation $\mathcal{R} : C \rightarrow C \rightarrow \mathcal{U}$, and a type $\overline{\mathcal{R}} = \sum_{a:C} \sum_{b:C} \mathcal{R} a b$, such that $\overline{\mathcal{R}}$ and $\alpha_{\overline{\mathcal{R}}} : \overline{\mathcal{R}} \rightarrow \mathbf{P}(\overline{\mathcal{R}})$ forms a **P**-coalgebra $\overline{\mathcal{R}}-\alpha_{\overline{\mathcal{R}}} : \text{Coalg}_S$, making the following diagram commute (where \Rightarrow are **P**-coalgebra morphisms).

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

Figure 2.5: TODO

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

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

Figure 2.6: TODO

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

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

$$R \equiv \equiv \quad (2.23)$$

meaning we can use the relation R , to show that two things of an \mathbf{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

$$p : R \rightarrow \equiv \quad (2.24)$$

$$q : \equiv \rightarrow R \quad (2.25)$$

and relations

$$\alpha : p \circ q \equiv \text{id}_{\equiv} \quad (2.26)$$

$$\beta : q \circ p \equiv \text{id}_R \quad (2.27)$$

Complete the construction of equality from any bisimulation relation over an \mathbf{M} -type

2.3.1 Bisimulation of Streams

TODO

2.3.2 Bisimulation of Delay monad

TODO

2.3.3 Bisimulation of ITrees

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

$$\frac{a, b : \mathcal{R} \quad a \equiv_R b}{\text{Ret } a \cong \text{Ret } b} \text{EqRet} \quad (2.28)$$

$$\frac{t, u : \text{itree } E \quad t \cong u}{\text{Tau } t \cong \text{Tau } u} \text{EqTau} \quad (2.29)$$

$$\frac{A : \mathcal{U} \quad e : \mathbf{E} \ A \quad k_1, k_2 : A \rightarrow \mathbf{itree} \ \mathbf{E} \ R \quad t \cong u}{\mathbf{Vis} \ e \ k_1 \cong \mathbf{Tau} \ e \ k_2} \mathbf{EqVis} \quad (2.30)$$

Now we just need to define $\alpha_{\overline{R}}$. 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 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

2.5 Closure properties of M-types

We define the product of two containers

$$(A, B) \times (C, D) \equiv (A \times C, \lambda(a, c), Ba \times Dc), \quad (2.31)$$

we can lift this rule, through the following diagram, used to define M-types

We now prove that

$$\mathbf{P}_{(A,B)}^n \mathbf{1} \times \mathbf{P}_{(C,D)}^n \mathbf{1} \equiv \mathbf{P}_{(A,B) \times (C,D)}^n \mathbf{1}, \quad (2.32)$$

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

$$\mathbf{P}_{(A,B)}^m \mathbf{1} \times \mathbf{P}_{(C,D)}^m \mathbf{1} \equiv \mathbf{P}_{(A,B) \times (C,D)}^m \mathbf{1}, \quad (2.33)$$

and show

$$\mathbf{P}_{(A,B)}^{m+1} \mathbf{1} \times \mathbf{P}_{(C,D)}^{m+1} \mathbf{1} \quad (2.34)$$

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

$$\equiv \sum_{a:A} B \ a \rightarrow \mathbf{P}_{(A,B)}^m \mathbf{1} \times \sum_{c:C} D \ c \rightarrow \mathbf{P}_{(C,D)}^m \mathbf{1} \quad (2.36)$$

$$\equiv \sum_{a,c:A \times C} (B \ a \rightarrow \mathbf{P}_{(A,B)}^m \mathbf{1}) \times (D \ c \rightarrow \mathbf{P}_{(C,D)}^m \mathbf{1}) \quad (2.37)$$

$$\equiv \sum_{a,c:A \times C} B \ a \times D \ c \rightarrow \mathbf{P}_{(A,B)}^m \mathbf{1} \times \mathbf{P}_{(C,D)}^m \mathbf{1} \quad (2.38)$$

$$\equiv \sum_{a,c:A \times C} B \ a \times D \ c \rightarrow \mathbf{P}_{(A,B) \times (C,D)}^m \mathbf{1} \quad (2.39)$$

$$\equiv \mathbf{P}_{(A,B) \times (C,D)}(\mathbf{P}_{(A,B) \times (C,D)}^m \mathbf{1}) \quad (2.40)$$

$$\equiv \mathbf{P}_{(A,B) \times (C,D)}^{m+1} \mathbf{1} \quad (2.41)$$

taking the limit we get

$$M_{(A,B)} \times M_{(C,D)} \equiv M_{(A,B) \times (C,D)} \quad (2.42)$$

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

$$\mathbf{stream} \ A \times \mathbf{stream} \ B \equiv M_{(A, \lambda _ , 1) \times (B, \lambda _ , 1)} \equiv \mathbf{stream} \ (A \times B) \quad (2.43)$$

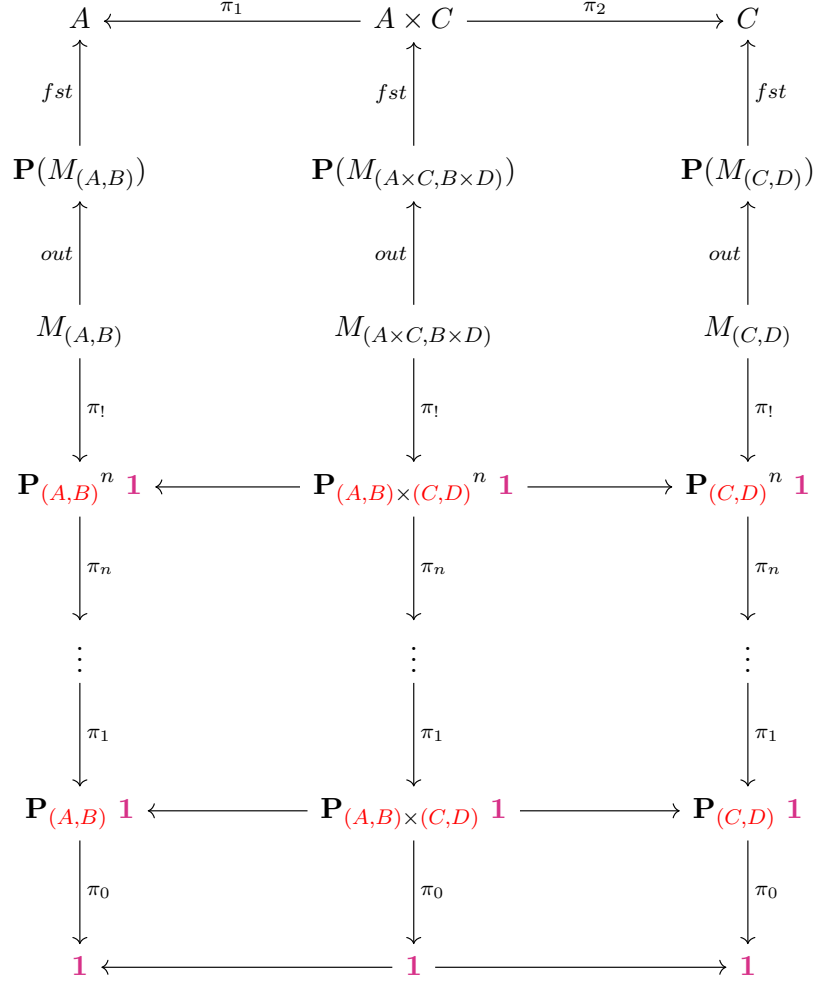


Figure 2.7: TODO

as expected, transporting along this gives us the definition for zip. More precisely we define zip as:

We start by defining the equality,

$$\begin{aligned}
 & W(A, B) (n+1) \times W(C, D) (n+1) \\
 \equiv & \left(\sum_{(a:A)} B a \rightarrow W(A, B) n \right) \times \left(\sum_{(c:C)} D c \rightarrow W(C, D) n \right) \\
 \equiv & \sum_{((a,c):A \times C)} (B a \rightarrow W(A, B) n) \times (D c \rightarrow W(C, D) n)
 \end{aligned} \tag{2.44}$$

by the two functions

$$f : ((a, c), (b, d)) \rightarrow ((a, b), (c, d)) \tag{2.45}$$

$$g : ((a, b), (c, d)) \rightarrow ((a, c), (b, d)) \tag{2.46}$$

such that $f \circ g \equiv id$ and $g \circ f \equiv id$ by **refl**.

$$zip = \mathbf{transport}_{(2.44)} \square (\sum_{(a,c):refl} ? \square \mathbf{cong} (\lambda x, Ba \times Dc \rightarrow x)((??))) \quad (2.47)$$

transporting over an equality $A \equiv B$, given by $f : A \rightarrow B$ and $g : B \rightarrow A$, is the same as applying the function f . The definition of zip therefore reduce do

$$zip = f \circ (\mathbf{transport}_{\sum_{(a,c):refl} ? \square \mathbf{cong} (\lambda x, Ba \times Dc \rightarrow x)((??))}) \quad (2.48)$$

2.5.1 Closure under products

The product of two M-types is again an M-type

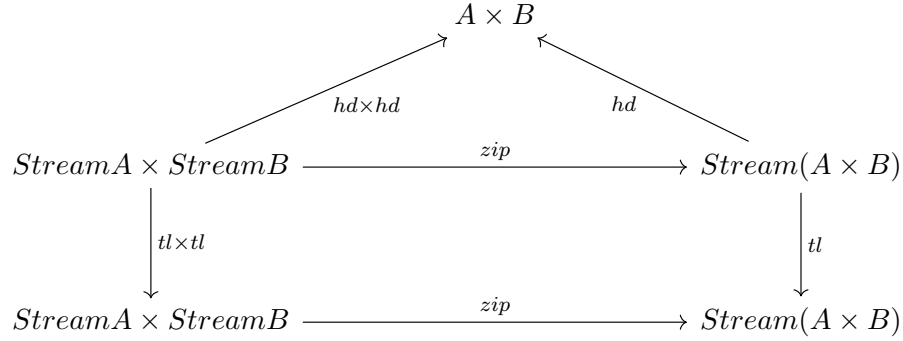


Figure 2.8: TODO

From this we get the computation rules

$$hd \times hd \equiv hd \circ zip \quad (2.49)$$

$$zip \circ tl \times tl \equiv tl \circ zip \quad (2.50)$$

another way to define the zip function is more directly, using the following lifting property of M-types

$$\mathbf{lift}_M \left(x : \prod_{n:\mathbb{N}} (A \rightarrow \mathbf{P}_S^n \mathbf{1}) \right) \left(u : \prod_{n:\mathbb{N}} (A \rightarrow \pi_n(x_{n+1}a) \equiv x_n a) \right) (a : A) : M S := (\lambda n, x \ n \ a), (\lambda n \ i, p \ n \ a \ i). \quad (2.51)$$

We can then define zip_{lift} as

$$zip_X \ n \ (x, y) = \begin{cases} \mathbf{1} & \text{if } n = 0 \\ (\mathbf{hd} \ x, \mathbf{hd} \ y), (\lambda _, zip_X \ m \ (\mathbf{tl} \ x, \mathbf{tl} \ y)), & \text{if } n = m + 1 \end{cases} \quad (2.52)$$

$$zip_\pi \ n \ (x, y) = \begin{cases} \mathbf{refl} & \text{if } n = 0 \\ \lambda i, (\mathbf{hd} \ x, \mathbf{hd} \ y), (\lambda _, zip_\pi \ m \ (\mathbf{tl} \ x, \mathbf{tl} \ y) \ i), & \text{if } n = m + 1 \end{cases} \quad (2.53)$$

$$zip_{lift} \ (x, y) := \mathbf{lift}_M \ zip_X \ zip \ (x, y) \quad (2.54)$$

Equality of Zip Definitions

We would expect that the two definitions for zip are equal

$$\mathbf{transport}_? a \equiv \mathbf{zip}_{lift} a \quad (2.55)$$

$$\equiv \mathbf{lift}_M \mathbf{zip}_X \mathbf{zip}_\pi (x, y) \quad (2.56)$$

$$\equiv (\lambda n, \mathbf{zip}_X n (x, y)), (\lambda n i, \mathbf{zip}_\pi n (x, y) i) \quad (2.57)$$

zero case X

$$\mathbf{zip}_X 0 (x, y) \equiv \mathbf{1} \quad (2.58)$$

Successor case X

$$\mathbf{zip}_X (m + 1) (x, y) \equiv (\mathbf{hd} x, \mathbf{hd} y), (\lambda _, \mathbf{zip}_X m (\mathbf{tl} x, \mathbf{tl} y)) \quad (2.59)$$

$$\equiv (\mathbf{hd} x, \mathbf{hd} y), (\lambda _, ? (\mathbf{tl} a)) \quad (2.60)$$

$$\equiv (\mathbf{hd} (\mathbf{transport}_? a)), (\lambda _, \mathbf{transport}_? (\mathbf{tl} a)) \quad (2.61)$$

$$\equiv \mathbf{transport}_? a \quad (2.62)$$

$$(2.63)$$

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

$$\equiv (), (\lambda i, \mathbf{zip}_\pi 0 (x, y) i) \quad (2.64)$$

$$\equiv \mathbf{1}, \mathbf{refl} \quad (2.65)$$

$$(2.66)$$

successor case

$$\equiv (\mathbf{zip}_X (m + 1) (x, y)), (\lambda i, \mathbf{zip}_\pi (m + 1) (x, y) i) \quad (2.67)$$

$$\equiv ((\mathbf{hd} x, \mathbf{hd} y), (\lambda _, \mathbf{zip}_X m (\mathbf{tl} x, \mathbf{tl} y))), (\lambda i, (\mathbf{hd} x, \mathbf{hd} y), (\lambda _, \mathbf{zip}_\pi m (\mathbf{tl} x, \mathbf{tl} y) i)) \quad (2.68)$$

$$(2.69)$$

2.5.2 Examples of fixed points

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

$$\mathbf{const}_X (n : \mathbb{N}) (c : \mathbb{N}) := \begin{cases} \mathbf{1} & n = 0 \\ (c, \lambda _, \mathbf{const}_X m c) & n = m + 1 \end{cases} \quad (2.70)$$

$$\mathbf{const}_\pi (n : \mathbb{N}) (c : \mathbb{N}) := \begin{cases} \mathbf{refl} & n = 0 \\ \lambda i, (c, \lambda _, \mathbf{const}_\pi m c i) & n = m + 1 \end{cases} \quad (2.71)$$

to get the definition of zero stream, we do

$$\mathbf{zeros} := \mathbf{lift}_M \mathbf{const}_X \mathbf{const}_\pi 0 \quad (2.72)$$

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

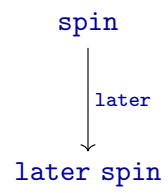


Figure 2.9: TODO

Chapter 3

Additions to the Cubical Agda Library

3.1 Lemma 10

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

$$U \equiv \sum_{f:C \rightarrow \mathcal{L}} \text{out} \circ f \equiv \text{step } f \quad (3.1)$$

$$\equiv \sum_{f:C \rightarrow \mathcal{L}} \text{in} \circ \text{out} \circ f \equiv \text{in} \circ \text{step } f \quad (3.2)$$

$$\equiv \sum_{f:C \rightarrow \mathcal{L}} f \equiv \text{in} \circ \text{step } f \quad (3.3)$$

$$\equiv \sum_{f:C \rightarrow \mathcal{L}} f \equiv \Psi f \quad (3.4)$$

$$\equiv \sum_{c:\text{Cone}} e \ c \equiv \Psi (e \ c) \quad (3.5)$$

$$\equiv \sum_{c:\text{Cone}} e \ c \equiv e \ (\phi \ c) \quad (3.6)$$

$$\equiv \sum_{c:\text{Cone}} c \equiv \phi \ c \quad (3.7)$$

$$\equiv \sum_{(u,q):\text{Cone}} (u, q) \equiv (\phi_0 \ u, \phi_1 \ u \ q) \quad (3.8)$$

$$\equiv \sum_{(u,q):\text{Cone}} \sum_{p:u \equiv \phi_0 \ u} q \equiv_{\lambda i, \text{Cone}_1(p \ i)} \phi_1 \ u \ q \quad (3.9)$$

$$\equiv \sum_{(u,p):\sum_{u:\text{Cone}_0} u \equiv \phi_0 \ u} \sum_{q:\text{Cone}_1 \ u} q \equiv_{\lambda i, \text{Cone}_1(p \ i)} \phi_1 \ u \ q \quad (3.10)$$

$$\equiv \sum_{q:\text{Cone}_1 u_0} q \equiv_{\lambda i, \text{Cone}_1(\text{funExt } p_0 \ i)} \phi_1 \ u_0 \ q \quad (3.11)$$

$$\vdots \quad (3.12)$$

$$\equiv \textcolor{red}{1} \quad (3.13)$$

3.2 Lemma 13

$$\sum_{((a,q):\sum_{a:\mathbb{N}\rightarrow A}\prod_{n:\mathbb{N}}a_{n+1}\equiv a_n)} \sum_{(u:\prod_{n:\mathbb{N}}Ba_n\rightarrow X_n)} \prod_{(n:\mathbb{N})} \pi \circ u_{n+1} \equiv_{\lambda i, B(q_n \ i)\rightarrow X_n} u_n \quad (3.14)$$

$$\equiv \sum_{a:A} \sum_{u:\prod_{n:\mathbb{N}}Ba\rightarrow X_n} \prod_{n:\mathbb{N}} \pi \circ u_{n+1} \equiv u_n \quad (3.15)$$

$$\equiv \sum_{a:A} \sum_{u:\prod_{n:\mathbb{N}}Ba\rightarrow X_n} \prod_{n:\mathbb{N}} \pi \circ u_{n+1} \equiv u_n \quad (3.16)$$

3.3 Missing postulates

Combine

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

$$\sum_{x_0:X_0} \sum_{y:\prod_{n:\mathbb{N}} X_{n+1}} (p \ y_0 \ x_0) \times \left(\prod_{n:\mathbb{N}} p \ y_{n+1} \ y_n \right) \quad (3.17)$$

$$\equiv \sum_{x_0:X_0} \sum_{y:\prod_{n:\mathbb{N}} X_{n+1}} (p \ y_0 \ x_0) \times \left(\prod_{n:\mathbb{N}} p \ y_{n+1} \ y_n \right) \quad (3.18)$$

$$\equiv \sum_{x:\prod_{n:\mathbb{N}} X_n} (p \ x_1 \ x_0) \times \left(\prod_{n:\mathbb{N}} p \ x_{n+2} \ x_{n+1} \right) \quad (3.19)$$

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

