M-types and Coinduction in HoTT and Cubical Type Theory

Lasse Letager Hansen, 201912345

Master's Thesis, Computer Science

March 15, 2020

Advisor: Bas Spitters



Abstract

in English...

Resumé

in Danish...

${\bf Acknowledgments}$

Lasse Letager Hansen, Aarhus, March 15, 2020.

Contents

Al	stract	iii
Re	sumé	v
Acknowledgments		vii
1	Introduction	1
2	M-types	3
	2.1 Containers / Signatures	3
	2.2 ITrees as M-types	
	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	
	3.2 Lemma 13	
	3.3 Missing postulates	15
4	Conclusion	17
Bi	oliography	19
Α	The Technical Details	21



Introduction

motivate and explain the problem to be addressed

example of a citation: [1]

get your bibtex entries from https://dblp.org/

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

$$\mathbf{P}_{S}: \mathcal{U} \to \mathcal{U}$$

$$\mathbf{P}(X) := \mathbf{P}_{S}(X) = \sum_{a:A} B(a) \to X$$
(2.1)

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

$$\mathbf{P}f: \mathbf{P}X \to \mathbf{P}Y$$

$$\mathbf{P}f(a,g) = (a, f \circ g)$$
(2.2)

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

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

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

$$\mathbf{P}_{S}(X) = A \times X \tag{2.4}$$

Now we define the coalgebra for this functor with type

$$Coalg_S = \sum_{CM} C \to \mathbf{P}C \tag{2.5}$$

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

$$_\Rightarrow_: \mathtt{Coalg}_S \to \mathtt{Coalg}_S$$

$$C - \gamma \Rightarrow D - \delta = \sum_{f:C \to D} \delta \circ f = \mathbf{P}f \circ \gamma$$

$$(2.6)$$

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

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

that is $\prod_{(C-\gamma: \mathtt{Coalg}_S)} \mathtt{isContr}(C-\gamma \Rightarrow \mathtt{M}_S-\mathtt{out})$. 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.

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

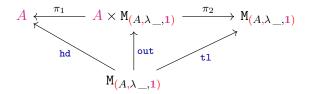


Figure 2.1: M-types of streams

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

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

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

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

$$(2.10)$$

2.2 ITrees as M-types

We want the following rules for ITrees

$$\frac{r:R}{\text{Ret }r:\text{itree E }R} \text{ I}_{\text{Ret}} \tag{2.11}$$

$$\frac{A: \mathcal{U} \quad a: \mathsf{E} \ A \quad f: A \to \mathsf{itree} \ \mathsf{E} \ R}{\mathsf{Vis} \ a \ f: \mathsf{itree} \ \mathsf{E} \ R} \ \mathsf{I}_{\mathsf{Vis}}. \tag{2.12}$$

Elimination rules

$$\frac{t: \text{itree E } R}{\text{Tau } t: \text{itree E } R} \text{ E}_{\text{Tau}}. \tag{2.13}$$

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

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

which is equal to

$$\mathbf{P}_{\mathbf{S}}(X) = X + R \times (\mathbf{0} \to X). \tag{2.15}$$

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

$$\mathbf{P}_{\mathbf{S}}(X) = X + R \tag{2.16}$$

meaning we get the following diagram.

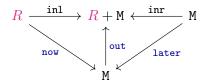


Figure 2.2: Delay monad

Meaning we can define the operations now and later using $in = out^{-1}$ 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

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

This will give us the polynomial functor

$$\mathbf{P}_{S}(X) = \sum_{x:R+\sum_{A:U} \in A} \lambda \{ \text{inl} _ \to \mathbf{0} ; \text{ inr } (A,e) \to A \} x \to X, \tag{2.18}$$

which simplifies to

$$\mathbf{P}_{S}(X) = (R \times (\mathbf{0} \to X)) + (\sum_{A:\mathcal{U}} \mathbf{E} \ A \times (A \to X)), \tag{2.19}$$

and further

$$\mathbf{P}_{S}(X) = R + \sum_{A:\mathcal{U}} \mathbf{E} \ A \times (A \to X). \tag{2.20}$$

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

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

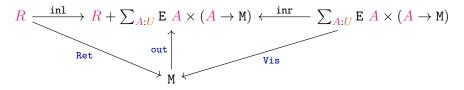


Figure 2.3: TODO

2.2.3 ITrees

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

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

Such that the (reduced) polynomial functor becomes

$$\mathbf{P}_{\mathbf{S}}(X) = X + R + \sum_{A:\mathcal{U}} ((\mathbf{E}\ A) \times (A \to X)) \tag{2.22}$$

Giving us the diagram

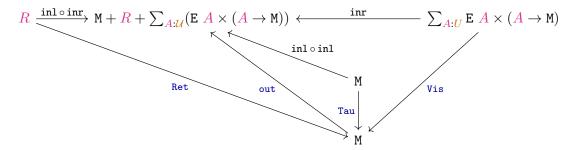


Figure 2.4: TODO

2.3 Co-induction Principle for M-types

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

$$C-\gamma \stackrel{\pi_1^{\overline{R}}}{\longleftarrow} \overline{R}-\alpha_{\overline{R}} \stackrel{\pi_2^{\overline{R}}}{\longrightarrow} C-\gamma$$

Figure 2.5: TODO

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

$$\operatorname{M-out} \stackrel{\pi_1^{\overline{\mathcal{R}}}}{\longleftarrow} \overline{\mathcal{R}} - \alpha_{\overline{\mathcal{R}}} \stackrel{\pi_2^{\overline{\mathcal{R}}}}{\longrightarrow} \operatorname{M-out}$$

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

$$R \equiv \equiv \tag{2.23}$$

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

$$p:R\to \equiv \tag{2.24}$$

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

and relations

$$\alpha: p \circ q \equiv \mathsf{id}_{\equiv} \tag{2.26}$$

$$\beta: q \circ p \equiv \mathrm{id}_R \tag{2.27}$$

Complete the construction of equality from any bisimulation relation over an 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.

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

$$\frac{t,u: \mathtt{itree} \ \mathtt{E} \ \frac{R}{R} \quad t \cong u}{\mathtt{Tau} \ t \cong \mathtt{Tau} \ u} \ \mathtt{EqTau} \tag{2.29}$$

$$\frac{A: \mathcal{U} \quad e: \mathbf{E} \quad A \quad k_1, k_2: A \to \mathsf{itree} \; \mathbf{E} \; R \quad t \cong u}{\mathsf{Vis} \; e \; k_1 \cong \mathsf{Tau} \; e \; k_2} \; \mathsf{EqVis}$$
 (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.

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

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

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

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

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

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

and show

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

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

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

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

$$\equiv \sum_{a,c,A\times C} B \ a \times D \ c \to \mathbf{P_{(A,B)}}^m \ \mathbf{1} \times \mathbf{P_{(C,D)}}^m \ \mathbf{1}$$
 (2.38)

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

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

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

taking the limit we get

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

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

$$stream \ A \times stream \ B \equiv M_{(A,\lambda)} = stream \ (A \times B)$$
 (2.43)

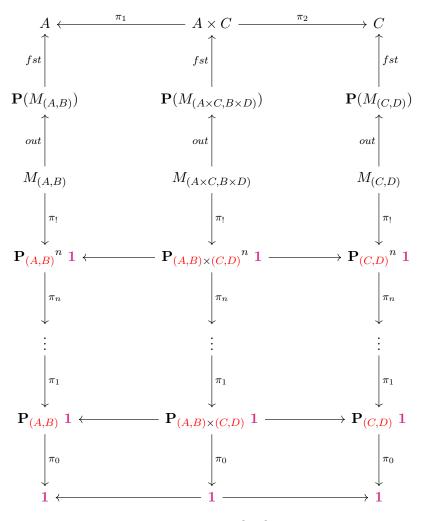


Figure 2.7: TODO

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

We start by defining the equality,

$$W(A,B)(n+1) \times W(C,D)(n+1)$$

$$\equiv \left(\sum_{(a:A)} B \ a \to W(A,B) \ n\right) \times \left(\sum_{(c:C)} D \ c \to W(C,D) \ n\right)$$

$$\equiv \sum_{((a,c):A \times C)} (B \ a \to W(A,B) \ n) \times (D \ c \to W(C,D) \ n)$$
(2.44)

by the two functions

$$f: ((a,c),(b,d)) \to ((a,b),(c,d))$$
 (2.45)

$$g: ((a,b),(c,d)) \to ((a,c),(b,d))$$
 (2.46)

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

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

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

$$zip = f \circ (\mathsf{transport}_{\sum_{(a,c):refl}? \ \Box cong \ (\lambda x, Ba \times Dc \to x)((??))}) \tag{2.48}$$

2.5.1 Closure under products

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

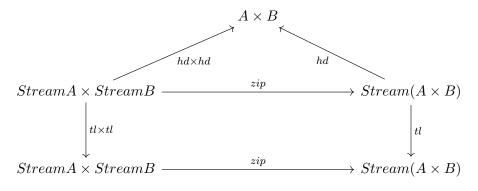


Figure 2.8: TODO

From this we get the computation rules

$$hd \times hd \equiv hd \circ zip \tag{2.49}$$

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

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

$$\operatorname{lift}_{\mathsf{M}} \left(x : \prod_{n:\mathbb{N}} (A \to \mathbf{P}_{\boldsymbol{S}}^{n} \mathbf{1}) \right) \left(u : \prod_{n:\mathbb{N}} (A \to \pi_{n}(x_{n+1}a) \equiv x_{n}a) \right) (a : A) : \mathsf{M} \, \boldsymbol{S} := (2.51)$$

$$(\lambda \, n, x \, n \, a), (\lambda \, n \, i, p \, n \, a \, i).$$

We can then define zip_{lift} as

$$\operatorname{zip}_X n (x,y) = \begin{cases} \mathbf{1} & \text{if } n = 0\\ (\operatorname{hd} x, \operatorname{hd} y), (\lambda_{_}, \operatorname{zip}_X m (\operatorname{tl} x, \operatorname{tl} y)), & \text{if } n = m+1 \end{cases} \tag{2.52}$$

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

$$zip_{lift}(x,y) := lift_{M} zip_{X} zip(x,y)$$
(2.54)

We would expect that the two definitions for zip are equal

transport?
$$a \equiv zip_{lift} a$$
 (2.55)

$$\equiv \text{lift}_{M} \ \text{zip}_{X} \ \text{zip}_{\pi} \ (x, y) \tag{2.56}$$

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

zero case X

$$zip_X \ 0 \ (x,y) \equiv 1 \tag{2.58}$$

Sucessor case X

$$\mathtt{zip}_X\ (m+1)\ (x,y) \equiv (\mathtt{hd}\ x,\mathtt{hd}\ y), (\lambda_,\mathtt{zip}_X\ m\ (\mathtt{tl}\ x,\mathtt{tl}\ y)) \tag{2.59}$$

$$\equiv (\operatorname{hd} x, \operatorname{hd} y), (\lambda_{-}, ? (\operatorname{tl} a)) \tag{2.60}$$

$$\equiv (hd (transport_{?}a)), (\lambda_{_}, transport_{?}(tl a))$$
 (2.61)

$$\equiv transport_?a$$
 (2.62)

(2.63)

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

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

$$\equiv 1, refl \tag{2.65}$$

(2.66)

successor case

$$\equiv (zip_X (m+1) (x,y)), (\lambda i, zip_\pi (m+1) (x,y) i)$$
(2.67)

$$\equiv ((\operatorname{hd} x, \operatorname{hd} y), (\lambda_{-}, \operatorname{zip}_{X} m \ (\operatorname{tl} x, \operatorname{tl} y))), (\lambda i, (\operatorname{hd} x, \operatorname{hd} y), (\lambda_{-}, \operatorname{zip}_{\pi} m \ (\operatorname{tl} x, \operatorname{tl} y) \ i)) \tag{2.68}$$

(2.69)

2.5.2 Examples of fixed points

We want to define spin, as being the fixed point spin = later spin, so that is again a final coalgebra, but of a M-type (which is a final coalgebra)



Figure 2.9: TODO

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

Additions to the Cubical Agda Library

3.1 Lemma 10

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

$$U \equiv \sum_{f:C \to \mathcal{L}} \operatorname{out} \circ f \equiv \operatorname{step} f \tag{3.1}$$

$$\equiv \sum_{f:C\to\mathcal{L}} \operatorname{in}\circ\operatorname{out}\circ f \equiv \operatorname{in}\circ\operatorname{step} f \tag{3.2}$$

$$\equiv \sum_{f:C\to\mathcal{L}} f \equiv \text{in} \circ \text{step } f \tag{3.3}$$

$$\equiv \sum_{f:C\to\mathcal{L}} f \equiv \Psi f \tag{3.4}$$

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

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

$$\equiv \sum_{c:Cone} c \equiv \phi \ c \tag{3.7}$$

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

$$\equiv \sum_{(u,q):Cone} \sum_{p:u \equiv \phi_0} q \equiv_{\lambda i,Cone_1(p\ i)} \phi_1 \ u \ q \tag{3.9}$$

$$\equiv \sum_{(u,p):\sum_{u:Cone_0} u \equiv \phi_0} \sum_{u \ q:Cone_1 \ u} q \equiv_{\lambda i,Cone_1(p \ i)} \phi_1 \ u \ q \tag{3.10}$$

$$\equiv \sum_{q:Cone_1u_0} q \equiv_{\lambda i,Cone_1(funExt\ p_0\ i)} \phi_1\ u_0\ q \tag{3.11}$$

$$\vdots (3.12)$$

$$\equiv 1 \tag{3.13}$$

3.2 Lemma 13

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

$$\equiv \sum_{a:A} \sum_{v:\Pi} \sum_{v:Ba \to X_n} \prod_{n:\mathbb{N}} \pi \circ u_{n+1} \equiv u_n \tag{3.15}$$

$$\equiv \sum_{a:A} \sum_{u:\prod_{n:\mathbb{N}} Ba \to X_n} \prod_{n:\mathbb{N}} \pi \circ u_{n+1} \equiv u_n \tag{3.16}$$

3.3 Missing postulates

Combine

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

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

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

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

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