Impredicative Encodings of Inductive and Coinductive Types

Steven Bronsveld
iCIS, Radboud University Nijmegen, The Netherlands

Herman Geuvers
iCIS, Radboud University Nijmegen, The Netherlands
Technical University Eindhoven, The Netherlands

Niels van der Weide
iCIS, Radboud University Nijmegen, The Netherlands

Abstract

In impredicative type theory (System F, also known as $\lambda 2$), it is possible to define inductive data types, such as natural numbers and lists. It is also possible to define coinductive data types such as streams. They work well in the sense that their (co)recursion principles obey the expected computation rules (the β-rules). Unfortunately, they do not yield a (co)induction principle [13, 32], because the necessary uniqueness principles are missing (the η-rules). Awodey, Frey, and Speight [4] used an extension of the Calculus of Constructions [9] (λC) with Σ-types, identity-types, and functional extensionality to define System F style inductive types with an induction principle, by encoding them as a well-chosen subtype, making them initial algebras.

In this paper, we extend their results to coinductive data types, and we detail the example of the stream data type with the desired coinduction principle (also called bisimulation). To do that, we first define quotient types (with the desired η-rules) and we also need a stronger form of the definable existential types. We also show that we can use the original method by Awodey, Frey and Speight for general inductive types by defining W-types with an induction principle. The dual approach for streams can be extended to M-types, the generic notion of coinductive types, and the dual of W-types.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Formulas-as-types, impredicativity, inductive types, coinductive types

Digital Object Identifier 10.4230/LIPIcs.FSCD.2025.11

Related Version This paper was based on the Master Thesis of Steven Bronsveld.
Master Thesis: `https://arxiv.org/abs/2505.13495`

Funding *Niels van der Weide*: This research was supported by the NWO project "The Power of Equality" OCENW.M20.380, which is financed by the Dutch Research Council (NWO).

Acknowledgements We thank Benno van den Berg and Daniel Otten for fruitful discussions. We thank the referees for their valuable comments and suggestions.

1 Introduction

In type theory based proof assistants, like Rocq [34] or Lean [11] or Agda [7], one can define functions and prove properties about them, so they form an integrated system for programming and proving. This is due to the combination of dependent types and inductive types, which, under the Curry-Howard formulas-as-types embedding, allow one to define inductive types with a recursion principle (to define functions by well-founded recursion) and an induction principle (to do proofs by induction).

In polymorphic type theory [16, 29] (System F, also known as $\lambda 2$), it is possible to create inductive data types, such as natural numbers and lists. It is also possible to create coinductive data types such as streams. They work well in the sense that their (co)recursion principles

obey the expected computation rules (the β-rules). In the Calculus of Constructions (λC), which extends polymorphic type theory with dependent types and higher order types, one can state induction principles for inductive data types and coinduction principles for coinductive data types. Unfortunately, these (co)induction principles are not provable in the type theory, because the necessary uniqueness principles are missing (the η-rules). Awodey, Frey, and Speight [4] use an extension of the Calculus of Constructions with Σ-types, equality-types and functional extensionality to provide System F style inductive types with an induction principle by encoding them as a subtype, making them initial algebras.

In this paper, we extend the results of Awodey, Frey, and Speight, to coinductive types and to general W-types. To deal with coinductive types, we first define a quotient type that has the desired induction principle. Then we take the quotient of the (System F) definable coinductive data type with an appropriate equivalence relation. We show that the type obtained is a final coalgebra and thus satisfies the expected coinduction principle, stating that bisimilarity and equality coincide. We show this in detail for the well-known type of infinite streams. To emphasize that the approach is general and that we can use the technique for general (co)inductive types, we show that we can define W-types as initial algebras, with an induction principle, and dually, we can define M-types [36] as final coalgebras with a coinduction principle. Categorically, initiality (finality) of an (co)algebra can be stated in equivalent ways and we show that each of these can be used to define an initial (final) (co)algebra in type theory.

Contributions. The contributions of this paper are as follows:

- a construction of the type of streams as an impredicative encoding (Theorem 4.16);
- a construction of W-types as an impredicative encoding (Theorem 6.7);
- a construction of M-types as an impredicative encoding (Theorem 7.10).

We also prove suitable induction and bisimulation principles for them.

Related Work. Our work extends the work by Awodey, Frey, and Speight [4]. Compared to their paper, we consider a wider variety of types, and, in particular, coinductive types like streams and M-types. Another extension of their work was by Ripoll Echeveste [31], who looked at how to extend the construction by Awodey, Frey, and Speight to types with an arbitrary homotopy level.

Parametricity gives another way to fix impredicative encodings. More specifically, if one assumes an internal version of the axiom of parametricity, then one can prove the induction principle for impredicative encodings [3, 37, 38]. One can prove the consistency of such systems using PER models [20].

There are various ways to construct quotient types. We use impredicative encodings in this paper, but, assuming propositional resizing, one could also construct them via equivalence classes [35]. Li [21] shows that various quotients in type theory are definable using normalization functions.

There are also other ways to construct M-types. For instance, one can construct M-types using W-types in an extensional setting [1, 36]. One can also perform such constructions in intensional type theory [2], and even in such a way that computation rules hold definitionally [28]. In our paper, we do not postulate the existence of W-types, and we directly obtain computation rules that hold up to definitional equality without needing a refinement. Hermida and Jacobs give a general construction of W-types and M-types in the context of fibrations [17], and they show that final coalgebras can be constructed as quotients. Our argument is slightly different: whereas the construction by Hermida and Jacobs is based on the category of applicative bisimulations, we use weakly final coalgebras.

Overview. In Section 2 we describe the type theory that we need. In Section 3 we show how to define quotients in type theory with the appropriate universal property, which we use in Section 4 to define the coinductive type of streams as a final coalgebra with a coinduction principle. In Section 5 we discuss alternative ways to construct inductive types via impredicative encodings. In Sections 6 and 7 we generalize inductive types to arbitrary W-types and coinductive types to arbitrary M-types, respectively.

2 The type theory

The type system we work in is the Calculus of Constructions λC, extended with identity types and Σ-types, and function extensionality and uniqueness of identity proofs. We work with the impredicatively defined data-types that we know from polymorphic λ-calculus. For background we refer to [9, 16, 6, 23] and the full rules are given in the Appendix A. We assume familiarity with typing judgments such as $(\Gamma \vdash a : A)$ and β-reduction. We often drop the context Γ and write $(a : A)$.

The system we use is the same as the one of Awodey, Frey, and Speight [4] and detailed in the master thesis of Sam Speight [33]. The synopsis is that it is the Calculus of Constructions (so we have dependent type theory with an impredicative bottom universe $\mathcal{U}$) extended with Σ-types and identity types (equality types, or $=$-types), with η-rules for Π-types and Σ-types and with function extensionality and uniqueness of identity proofs (UIP). The system can also be seen as a subset of homotopy type theory (HoTT) [35] with one impredicative bottom universe. The relation with System F is often emphasized by various authors [4, 33, 31], as the impredicative encodings of data types is a crucial point.

As usual, there are two notions of equality. First of all we have definitional equality, generated from β, denoted by $t \overset{\beta}{=} q$, and η, denoted by $t \overset{\eta}{=} q$. Then there is propositional equality, which is a type in the system, given by the rules for the *identity type*, which we denote by $p : t =_A q$ for $t, q : A$ (p is a proof term of type $t =_A q$). We often omit the subscript A. If two terms t and q are definitionally equal, they are indistinguishable for the type system, i.e. we have $\mathtt{refl} : t =_A q$, where $\mathtt{refl}$ is the reflexivity proof-term.

We add the following two axioms. Let X be a type and Y be a type family on X.

$$\mathtt{FunExt} : \Pi(f, g : \Pi(x : X), Y\ x).(\Pi(x : X).f\ x = g\ x) \implies f = g$$

$$\mathtt{UIP} : \Pi(X : \mathcal{U}).\Pi(x, y : X).\Pi(p, q : x = y).p = q$$

We use Σ-types to create subtypes. The type $\Sigma(x : A).P\,x$ is seen as a subtype of A, consisting of pairs $\langle a, p\rangle$ where $p : P\,a$. It is in general not the case that $a = a' \Leftrightarrow \langle a, p\rangle = \langle a', q\rangle$. We only have this equivalence in case $P\,x$ is proof-irrelevant, that is, $P\,x$ is a type in which all inhabitants are equal. As we assume UIP, predicates $P\,x$ of the form $\ldots \to q = t$ will be proof-irrelevant, and in most (all) of the situations that occur in this paper, a subtype $\Sigma(x : A).P\,x$ will be of that form. Hence, we frequently use the following Lemma.

▶ **Lemma 2.1** (Sigma injection principle). *For $X : \mathcal{U}$ and $P : X \to \mathcal{U}$ with $P\ x$ proof-irrelevant (i.e. all terms of type $P\ x$ are equal), we have (where $\mathtt{pr1} : (\Sigma(x : X).P\ x) \to X$ denotes the first projection)*

$$\Pi(y, y' : \Sigma(x : X).P\ x).y = y' \iff \mathtt{pr1}\ y = \mathtt{pr1}\ y'$$

Proof. From left to right is immediate. For the right to left implication, we first prove $\Pi(x, x' : X).x = x' \to \Pi(z : P\ x).\Pi(z' : P\ x').\langle x, z\rangle = \langle x', z'\rangle$ using the elimination rule for the identity type (see Appendix A) and proof-irrelevance of $P\ x$. Then the Lemma follows from the η-rule for Σ-types. ◀

3 Quotients

To define final coalgebras impredicatively, we first need quotient types. A quotient type consists of terms of a given type $D : \mathcal{U}$ that are considered equivalent according to a relation $R : D \to D \to \mathcal{U}$. It is possible to define quotient types in impredicative type theory. Similar to natural numbers and lists, this definition satisfies the expected β-rule, but fails to satisfy the η-rule. We show that it is possible to extend the technique of [4] to define quotients with an η-rule. We use this improved quotient type to show that the class-function, that lifts an element of the base type to an element of the quotient type, is indeed a surjective function. In Section 4 we use the quotients as defined here to create stream types.

3.1 Impredicative quotient type

In the remainder of this section, we fix a type $D : \mathcal{U}$ and a relation $R : D \to D \to \mathcal{U}$. First, we define the predicate that states that a function behaves the same on all equivalence classes of a relation R.

▶ **Definition 3.1.** We say that a function f **respects** R if for all $x, y : D$ such that $R\ x\ y$, we have $f\ x = f\ y$. We write $\mathtt{EqCls}\ f\ R$ for the type expressing that f respects R.

▶ **Definition 3.2.** We define the **quotient type** $\mathtt{quot}^*\ D\ R$ to be $\Pi(C : \mathcal{U}).\Pi(f : D \to C).\mathtt{EqCls}\ f\ R \to C$. The **class function** $\mathtt{cls}^* : D \to \mathtt{quot}^*\ D\ R$ is defined to be

$$\mathtt{cls}^* := \lambda(d : D).\lambda(C : \mathcal{U}).\lambda(f : D \to C).\lambda(H : \mathtt{EqCls}\ f\ R).f\ d.$$

We sometimes write $D/^*R := \mathtt{quot}^*\ D\ R$.

We can lift a function $(f : D \to E)$ that respects R to a function $(\widehat{f} : \mathtt{quot}^*\ D\ R \to E)$. This lifting is done by the recursor for quotient types.

▶ **Definition 3.3.** We define the **recursor for quotient types** $\mathtt{rec}^*_\mathtt{q}$ as follows.

$$\mathtt{rec}^*_\mathtt{q} := \lambda(C : \mathcal{U}).\lambda(f : D \to C).\lambda(H : \mathtt{EqCls}\ f\ R).\lambda(q : \mathtt{quot}^*\ D\ R).q\ C\ f\ H$$

We usually write $\widehat{f} := (\lambda q.\ \mathtt{rec}^*_\mathtt{q}\ C\ f\ H\ q)$ if C and H are clear from the context.

The lifted function satisfies $\widehat{f} \circ \mathtt{cls}^* = f$, which is the β-rule for quotient types. This is stated in the following lemma, which is proved by β-reduction.

▶ **Lemma 3.4.** *Given $f : D \to C$ and $H : (\mathtt{EqCls}\ f\ R)$ we have that $(\mathtt{rec}^*_\mathtt{q}\ C\ f\ H) \circ \mathtt{cls}^* = f$. This means that the following diagram commutes.*

$$\begin{array}{ccc} D & \xrightarrow{\mathtt{cls}^*} & D/^*R \\ {\scriptstyle f}\downarrow & \swarrow_{=} & \\ C & & \end{array} \qquad \widehat{f}$$

▶ **Lemma 3.5.** *If we have $R\ x\ y$, then we also have $\mathtt{cls}^*\ x = \mathtt{cls}^*\ y$.*

Proof. The statement follows by function extensionality: for $C : \mathcal{U}$, $f : D \to C$ and $H : (\mathtt{EqCls}\ f\ R)$ we prove $\mathtt{cls}^*\ x\ C\ f\ H = \mathtt{cls}^*\ y\ C\ f\ H$. This follows from $\mathtt{cls}^*\ x\ C\ f\ H \overset{\beta}{=} f\ x$ and the fact that $f\ x = f\ y$, which follows from $R\ x\ y$ by $(H : (\mathtt{EqCls}\ f\ R))$. ◀

▶ Note 3.6. In the definition of $\texttt{quot}^*\ D\ R$, we do not require R to be an equivalence relation. A function f that acts on elements of $\texttt{quot}^* D\ R$ needs to satisfy $\texttt{EqCls}\ f\ R$. Hence, $\texttt{quot}^* DR$ is the quotient of D by the smallest equivalence relation containing R.

We might expect that the function $\widehat{f}$ is the unique lifted function, and that any function h that satisfies the β-rule is equal to $\widehat{f}$. This uniqueness requirement or η-rule is unfortunately not satisfied[1]. We now take a look at a categorical definition of quotients. Using this definition and its uniqueness requirement, we can apply the same encoding technique as in [4] to create a quotient type that satisfies the uniqueness property. Categorically, we want quotients to satisfy the uniqueness rule in the left diagram of Figure 1.

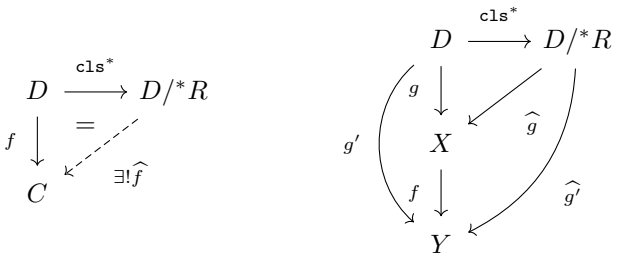


Figure 1 Uniqueness rule for quotients; Commutative diagram of the η-rule for quotient types.

We can give categorically equivalent definition of uniqueness. Suppose we have morphisms $(g : D \to X)$ and $(g' : D \to Y)$ that respect R, so we have $(\texttt{EqCls}\ g\ R)$ and $(\texttt{EqCls}\ g'\ R)$. By the β-rule, we know that $\widehat{g} \circ \texttt{cls}^* = g$ and $\widehat{g'} \circ \texttt{cls}^* = g'$. If we have a morphism $(f : X \to Y)$, such that $g' = f \circ g$, we want to have that $\widehat{g'} = f \circ \widehat{g}$ (the bottom right triangle in the diagram on the right in Figure 1). This is the η-rule for quotient types.

3.2 Impredicative encoding of quotients with η

We now define an impredicative encoding of quotients that satisfies the η-rule. We basically follow the same approach as for the natural numbers in [4].

▶ **Definition 3.7.** We define the **inductive quotient type**, which we denote by $\texttt{quot}$, as $\Sigma(q : \texttt{quot}^*\ D\ R).\texttt{LimQuot}\ q$ where $\texttt{LimQuot}\ (q : \texttt{quot}^*\ D\ R)$ says that for all types $X, Y : \mathcal{U}$, functions $g : D \to X$ and $g' : D \to Y$ that respect R, and functions $f : X \to Y$, we have $f\ (\texttt{rec}^*_{\texttt{q}}\ X\ g\ H\ q) = \texttt{rec}^*_{\texttt{q}}\ Y\ g'\ H'\ q$ whenever $f \circ g = g'$. We define the **recursor** as follows.

$$\texttt{rec}_{\texttt{q}} := \lambda(C : U).\lambda(f : D \to C).\lambda(H : \texttt{EqCls}\ f\ R).\lambda(u : \texttt{quot}\ D\ R).\texttt{rec}^*_{\texttt{q}}\ C\ f\ H\ (\texttt{pr1}\ u)$$

We usually write $\overline{f} := (\lambda u.\texttt{rec}_{\texttt{q}}\ C\ f\ H\ u)$ if C and H are clear from context. We sometimes write $D/R := \texttt{quot}\ D\ R$.

To define the new $\texttt{cls}$ function, we need to show $\texttt{LimQuot}\ (\texttt{cls}^*\ d)$ for all $d : D$. This follows from β-reduction.

▶ **Lemma 3.8** ($\texttt{LimQuotCls}$[2])**.** *We have a term* $\mathit{LimQuotCls} : \prod(d : D).\mathit{LimQuot}\ (\mathit{cls}^*\ d)$.

[1] The technique in [13] can be used to create a counter model.
[2] When we make use of a witness of a statement, we name that witness after the theorem number.

▶ **Definition 3.9.** We define the **class function** $\texttt{cls}$ to be $\lambda(d : D).\langle \texttt{cls}^*\ d,\ \texttt{LimQuotCls}\ d\rangle$.

We still have that $\texttt{cls}\ x = \texttt{cls}\ y$ for all $x, y : D$ such that $R\ x\ y$.

▶ **Lemma 3.10** (EqCls2). *For all $x, y : D$ we have $\mathtt{cls}\ x = \mathtt{cls}\ y$ if $R\ x\ y$.*

Proof. From $R\ x\ y$ we get $\texttt{cls}^*\ x = \texttt{cls}^*\ y$ by Lemma 3.5. Now the results follows from

$$\texttt{cls}\ x = \langle \texttt{cls}^*\ x,\ \texttt{LimQuotCls}\ x\rangle = \langle \texttt{cls}^*\ y,\ \texttt{LimQuotCls}\ y\rangle = \texttt{cls}\ y,$$

where the middle equation follows from Lemma 2.1 and the fact that $\texttt{LimQuot}$ is a proof-irrelevant predicate, since it ends in an equality. ◀

We are now ready to prove the computation rule (β-rule) for the new quotient type $\texttt{quot}$.

▶ **Lemma 3.11.** *Given $(C : \mathcal{U})$, $(g : D \to C)$ and $(H : \mathtt{EqCls}\ g\ R)$ we have that $(\mathtt{rec_q}\ C\ g\ H) \circ \mathtt{cls} = g$.*

Proof. The proof follows from function extensionality and β-reduction. ◀

Next, we show that the identity case of the η-rule holds: $\overline{\texttt{cls}} = \texttt{id}_{D/R}$.

▶ **Lemma 3.12** (IdLift). *We have $\mathtt{rec_q}\ D/R\ \mathtt{cls}\ \mathtt{EqCls2} = \mathtt{id}_{D/R}$.*

Proof. The proof is by function extensionality and the fact that to prove an equality of terms of type $\texttt{quot}\ D\ R$ we only have to prove the equality of their first projections (Lemma 2.1). This means we have to prove that $\texttt{pr1}\ (\texttt{rec}_\texttt{q}\ D/R\ \texttt{cls}\ \texttt{EqCls2}\ u)\ C\ g\ H$ and $(\texttt{pr1}\ u)\ C\ g\ H$ are equal for all u, C, g, and H. This follows from $\texttt{LimQuot}\ (\texttt{pr1}\ u)$. ◀

Using the previous lemma, we show that the general η-rule is satisfied.

▶ **Theorem 3.13.** *Suppose that we have a type $(C : \mathcal{U})$ and a function $(g : D \to C)$ with $(H : (\mathtt{EqCls}\ g\ R))$. For every $(f : D/R \to C)$ with $f \circ \mathtt{cls} = g$, we have $f = \overline{g}$.*

Proof. Assume we have D, R, C, g, H and f as described. We want to show that $f = \texttt{rec}_\texttt{q}\ C\ g\ H$. Let $(\langle q, p\rangle : D/R)$ where $(q : \texttt{quot}^* D\ R)$ and $(p : \texttt{LimQuot}\ q)$. We have that $f \circ \texttt{cls} = g$ by assumption. Obviously, we have $(H' : \texttt{EqCls}\ (f \circ \texttt{cls})\ R)$. We thus have that all requirements of p are satisfied. The statement follows by function extensionality.

$$f\ \langle q, p\rangle \overset{\texttt{IdLift}}{=} f\ (\texttt{rec}_\texttt{q}\ D/R\ \texttt{cls}\ \texttt{EqCls2}\ \langle q, p\rangle) \overset{p}{=} \texttt{rec}_\texttt{q}\ C\ g\ H\ \langle q, p\rangle$$ ◀

We conclude this section by proving the induction principle, which says that whenever a proof-irrelevant property P holds for all $(\texttt{cls}\ (d : D))$, we also have that P holds for all $u : D/R$. The improved $\texttt{quot}$ type satisfies this property since it follows from the η-rule. We show this in the following theorem.

▶ **Theorem 3.14.** *For proof-irrelevant $(P : D/R \to \mathcal{U})$, if $\Pi(x : D).\ P\ (\mathtt{cls}\ x)$, then $\Pi(u : D/R).\ P\ u$.*

Proof. Let $(A : \Pi(x : D).\ P\ (\texttt{cls}\ x))$ and consider $g : D \to (\Sigma(u : D/R).P\ u)$ defined by $g(x) := \langle \texttt{cls}\ x, A\ x\rangle$. We will show that $\texttt{pr2} \circ \overline{g} : \Pi(u : D/R).\ P\ u$.

We have $R\ x\ y \implies \texttt{cls}\ x = \texttt{cls}\ y$ by Lemma 3.10 and so $g\ x = g\ y$ by Lemma 2.1. So we can lift g to $\overline{g}$ and we have $\overline{g} \circ \texttt{cls} = g$. So $\texttt{pr1} \circ \overline{g} \circ \texttt{cls} = \texttt{pr1} \circ g = \texttt{cls}$ and by Theorem 3.13 (taking $\texttt{pr1} \circ \overline{g}$ for f and $\texttt{cls}$ for g in the Theorem), we find $\texttt{pr1} \circ \overline{g} = \overline{\texttt{cls}} = \texttt{id}_{D/R}$ (where the last equality is Lemma 3.12). But then for $(u : D/R)$, we have $\texttt{pr2}\ (\overline{g}\ u) : P\ u$. We conclude that $\texttt{pr2} \circ \overline{g} : \Pi(u : D/R).\ P\ u$. ◀

4 Streams

We now treat the type of streams as an example of an impredicative encoding of a coinductive type. We show that this type satisfies the expected β-rules and the η-rule. Coinductive types are dual to inductive types and describe (possibly) infinite data structures. These streams, or infinite lists, shall range over some element type $(E : \mathcal{U})$. To construct streams, we use existential types, the type theoretical analog of the existential quantifier. We first give the wel-known impredicative definition of streams and show that the β-rules are satisfied. Next, we take a look at the categorical definition of coinductive streams as a final coalgebra, which we use to give an impredicative encoding of a stream type that also satisfies the η-rule and the coinduction principle.

4.1 Existential types

▶ **Definition 4.1.** We define the **impredicative existential type** $\exists^*(X : \mathcal{U}).P\,X$ to be $\Pi(Y : \mathcal{U}).(\Pi(X : \mathcal{U}).(P\,X) \to Y) \to Y$. The $\mathtt{pack}^*$ constructor is defined to be

$$\mathtt{pack}^* := \lambda(X : \mathcal{U}).\lambda(t : P\,X).\lambda(Y : \mathcal{U}).\lambda(k : \Pi(X : \mathcal{U}).(P\,X) \to Y).k\ X\ t$$

So $\mathtt{pack}^* : \Pi(X : \mathcal{U}).(P\,X) \to \exists^*(X : \mathcal{U}).P\,X$. We define the recursor $\mathtt{rec}^*_\exists$ to be

$$\mathtt{rec}^*_\exists := \lambda(Y : \mathcal{U}).\lambda(f : \Pi(Z : \mathcal{U}).(P\,Z) \to Y).\lambda(e : \exists^* X.P).e\ Y\ f$$

We denote these existential types using the shorthand notation $\exists^* X.P := \exists^*(X : \mathcal{U}).P\,X$.

The recursor satisfies the usual β-equality, which says that $\mathtt{rec}^*_\exists\ Y\ f\ (\mathtt{pack}^*\ X\ t) \overset{\beta}{=} f\ X\ t$. When we *unpack* an existential type $(e : \exists^* X.P)$ using the recursor $\mathtt{rec}^*_\exists$, we obtain values $(X : \mathcal{U})$ and $(t : P(X))$. These values are unrelated to the values we used to create the member e. This is by design. However, if we use the $\mathtt{pack}^*$ function, to re-package the X and t values $e' := \mathtt{pack}^*\ X\ t$, we want to have $e = e'$. In other words, if we unpack an $(e : \exists^* X.P)$, and then re-pack it, we would like it to be equal to e. To make sure that this is the case, we define $\exists X.P$ as a subtype of $\exists^* X.P$.

▶ **Definition 4.2.** We define the **existential type** $\exists(X : \mathcal{U}).P(X)$ to be $\Sigma(y : \exists^* X.P\,X).\mathtt{Lim}_\exists\ y$, where $\mathtt{Lim}_\exists(y : \exists^* X.P\,X)$ is defined to be $\exists^*(X : \mathcal{U}).\exists^*(t : P\,X).y = \mathtt{pack}^*\ X\ t$ [3] The constructor $\mathtt{pack}$ is defined to be

$$\mathtt{pack} := \lambda(X : \mathcal{U}).\lambda(t : P\,X).\langle \mathtt{pack}^*\ X\ t, \mathtt{pack}^*\ X\ (\mathtt{pack}^*\ t\ \mathtt{refl})\rangle$$

(Note that indeed $\mathtt{pack}^*\ X\ (\mathtt{pack}^*\ t\ \mathtt{refl}) : \exists^*(X' : \mathcal{U}).\exists^*(t' : P\,X').\mathtt{pack}^*\ X\ t = \mathtt{pack}^*\ X'\ t'.$) We define the recursor $\mathtt{rec}_\exists$ as follows.

$$\mathtt{rec}_\exists := \lambda(Y : \mathcal{U}).\lambda(f : \Pi(Z : \mathcal{U}).(P\,Z) \to Y).\lambda(e : \exists X.P).\mathtt{rec}^*_\exists\ Y\ f\ (\mathtt{pr1}\ e).$$

We denote these existential types using the shorthand notation $\exists X.P := \exists(X : \mathcal{U}).P\,X$.

Note that the definition above makes sense, because $\mathtt{pack}^*\ X\ t \overset{\beta}{=} \mathtt{pack}^*\ X\ t$, so $\mathtt{refl}$ is a proof of this equality. The expected β-equality still holds, so we have $\mathtt{rec}_\exists\ Y\ f\ (\mathtt{pack}\ X\ t) \overset{\beta}{=} f\ X\ t$. We give the main results about $\exists X.P$ that we need, especially in Lemma 4.12.

[3] Another way to define $\mathtt{Lim}_\exists$ is by using $\exists$-morphisms and requiring them to be unique, similar to quotients in Definition 3.7. This is equivalent, but the proofs of the Lemmas are slightly longer.

▶ **Lemma 4.3.** *For all* $e : \exists X.P$ *there are* $X : \mathcal{U}$ *and* $t : P\,X$ *such that* $e = \mathit{pack}\,X\,t$. *In addition, we have* $\mathit{rec}_\exists\ (\exists X.P)\ \mathit{pack} = \mathit{id}_{\exists X.P}$.

Proof. Given $e : \exists X.P$ we have $\mathtt{pr1}\,e : \exists^* X.P\,X$ and $\mathtt{pr2}\,e : \exists^* X.\exists^* t.\mathtt{pr1}\ e = \mathtt{pack}^*\ X\,t$. We use $\mathtt{pr2}\,e$ to eliminate the $\exists^*$ and find X and t for which $\mathtt{pr1}\ e = \mathtt{pack}^*\ X\,t$. For this X and t we have e equals $\langle \mathtt{pr1}\,e, \mathtt{pr2}\,e\rangle = \langle \mathtt{pack}^*\ X\,t, \mathtt{pack}^*\ X\ (\mathtt{pack}^*\ t\ \mathtt{refl})\rangle = \mathtt{pack}\,X\,t$. Finally, we prove that $\mathtt{rec}_\exists\ (\exists X.P)\ \mathtt{pack}\ e = e$ for all $e : \exists X.P$. From e we get X and t such that $e = \mathtt{pack}\,X\,t$, so $\mathtt{rec}_\exists\ (\exists X.P)\ \mathtt{pack}\ e = \mathtt{rec}_\exists\ (\exists X.P)\ \mathtt{pack}\ (\mathtt{pack}\,X\,t) = \mathtt{pack}\,X\,t = e$. ◀

4.2 Impredicative stream type

We first give the well-known impredicative definition of a stream type with its "head" and "tail" destructors and its corecursor. This definition satisfies the computation rules.

▶ **Definition 4.4.** We define the **impredicative stream type**, denoted by $\mathtt{Stream}^*$, to be $\exists(X : \mathcal{U}).X \times (X \to E) \times (X \to X)$. The destructor $\mathtt{hd}^*(s : \mathtt{Stream}^*)$ is defined as $\mathtt{rec}_\exists\ E\ (\lambda X.\lambda\langle x, h, t\rangle.\ h\ x)\ s$, and the destructor $\mathtt{tl}^*(s : \mathtt{Stream}^*)$ is

$$\mathtt{rec}_\exists\ E\ (\lambda X.\lambda\langle x, h, t\rangle.\ \mathtt{pack}\ X\ \langle t\ x, h, t\rangle)\ s\ :\ \mathtt{Stream}^*.$$

Finally, we define the **corecursor** $\mathtt{corec}^*_\mathtt{s}$ as follows.

$$\mathtt{corec}^*_\mathtt{s} := \lambda(X : \mathcal{U}).\lambda(h : X \to E).\lambda(t : X \to X).\lambda(x : X).\mathtt{pack}\ X\ \langle x, h, t\rangle.$$

It is a simple check that the introduction rule (the corecursor) and elimination rules (the head and tail functions) compute as one would expect.

$$\mathtt{hd}^*(\mathtt{corec}^*_\mathtt{s}\ X\ h\ t\ x) \overset{\beta}{=} h\ x \qquad\qquad \mathtt{tl}^*(\mathtt{corec}^*_\mathtt{s}\ X\ h\ t\ x) \overset{\beta}{=} \mathtt{corec}^*_\mathtt{s}\ X\ h\ t\ (t\ x)$$

In category theory, we define streams as the final coalgebra of the endofunctor $\mathcal{S}(X) := E \times X$. We have a category of $\mathcal{S}$-coalgebras denoted as $\mathcal{S}$-**CoAlg** with objects $\langle X, \langle h, t\rangle\rangle$ where X is a set and $(h : X \to E)$ and $(t : X \to X)$ are functions. Morphisms in $\mathcal{S}$-**CoAlg** satisfy the equations $h' \circ f = h$ and $t' \circ f = f \circ t$, expressed in Figure 2. The final object $\langle F, \langle \mathrm{hd}, \mathrm{tl}\rangle\rangle$ of the category of $\mathcal{S}$-coalgebras $\mathcal{S}$-**CoAlg** has exactly one morphism $(u_X : \langle X, \langle h, t\rangle\rangle \to \langle F, \langle \mathrm{hd}, \mathrm{tl}\rangle\rangle)$ for each coalgebra $\langle X, \langle h, t\rangle\rangle$. In formulas, this means that for each $\mathcal{S}$-**CoAlg** morphism $(f : X \to Y)$, we have that $u_Y \circ f = u_X$.

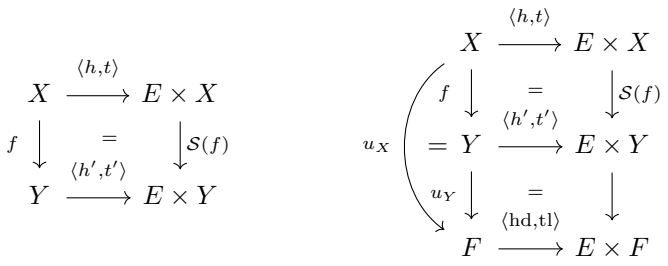


Figure 2 Diagram for $\mathcal{S}$-**CoAlg** morphisms (left) and for the final $\mathcal{S}$-coalgebra (right).

Next, we translate the categorical notion of final $\mathcal{S}$-coalgebra back to type theory. We define the predicate `MorphStream` that states that $f : X \to Y$ forms an $\mathcal{S}$-coalgebra morphism. This predicate mirrors the commutative diagram from the categorical definition in Figure 2 (left). For $(X, Y : \mathcal{U})$, $(h : X \to E)$, $(t : X \to X)$, $(h' : Y \to E)$, $(t' : Y \to Y)$ and $(f : X \to Y)$, we define `MorphStream` as follows.

$$\texttt{MorphStream}\ X\ h\ t \quad Y\ h'\ t' \quad f : X \to Y \quad := \quad h' \circ f = h \ \wedge\ t' \circ f = f \circ t$$

We want to define an encoding of $\texttt{Stream}^*$ that satisfies the η-rule, which is taken from the final coalgebra of $\mathcal{S}$ as shown in Figure 2 (right). Given a morphism $(f : X \to Y)$, we want that $u_Y \circ f = u_X$, where $(u_X : X \to F)$ and $(u_Y : Y \to F)$ are the morphisms created using the corecursor for $\texttt{Stream}^*$: $u_X := \texttt{corec}_s^*\ X\ h\ t$ and $u_Y := \texttt{corec}_s^*\ Y\ h'\ t'$.

In the case of quotients (and for natural numbers, see [4]), we created a subtype of $\texttt{quot}^*\ D\ R$, by pairing each element with a proof that it satisfies the η-rule. This made $\texttt{quot}\ D\ R$ a subtype of $\texttt{quot}^*\ D\ R$. In the case of streams, we take the dual notion of a subtype: a quotient. The quotient equates two streams that are related by $u_Y \circ f = u_X$. To define this quotient, we define a relation `CoLimStr` that relates streams σ and τ if they can be translated into each other by some $\mathcal{S}$-morphism f. The relation `CoLimStr` is given below.

▶ **Definition 4.5.** We define the relation `CoLimStr` between streams $(\sigma, \tau : \texttt{Stream}^*)$ where $(\texttt{CoLimStr}\ \sigma\ \tau)$ holds if σ and τ are related by some $\mathcal{S}$-coalgebra morphism.

$$\begin{aligned}\texttt{CoLimStr}\ \sigma\ \tau \quad := \quad & \exists(X, Y : \mathcal{U}).\exists^{(h:X\to E)}_{(h':Y\to E)}.\exists^{(t:X\to X)}_{(t':Y\to Y)}.\exists(f : X \to Y).\exists(x : X) \\ & (\texttt{MorphStream}\ X\ h\ t \quad Y\ h'\ t' \quad f) \ \wedge \\ & \sigma = \texttt{corec}_s^*\ X\ h\ t\ x \ \wedge\ \tau = \texttt{corec}_s^*\ Y\ h'\ t'\ (f\ x)\end{aligned}$$

We use the infix notation $(\sigma \equiv \tau)$ to denote that $(\texttt{CoLimStr}\ \sigma\ \tau)$ holds.

▶ Note 4.6. The relation `CoLimStr` is neither *symmetric* nor *transitive*. By Note 3.6, this does not hinder us since the equality relation on $(\texttt{cls}\ x)$ is an equivalence relation.

We now define the `Stream` type as a quotient and then we define the adapted "head" and "tail" function on this quotient type.

▶ **Definition 4.7.** We define the **coinductive stream type**, which we denote by `Stream`, as `quot Stream`* `CoLimStr`.

To define the new head and tail functions `hd` and `tl` as lifted functions of $\texttt{hd}^*$ and $\texttt{tl}^*$ we need to show that $\texttt{hd}^*$ and $\texttt{tl}^*$ are well-defined.

▶ **Lemma 4.8.** *We have* $\texttt{EqHd} : (\texttt{EqCls}\ \texttt{hd}^* \equiv)$*, the head function* $\texttt{hd}^*$ *is constant on all equivalence classes of* `CoLimStr` *and* $\texttt{EqTl} : (\texttt{EqCls}\ (\texttt{cls} \circ \texttt{tl}^*) \equiv)$*, the lifted tail function* $\texttt{cls} \circ \texttt{tl}^*$ *is constant on all equivalence classes of* `CoLimStr`*.*

$$\begin{aligned}&\Pi(\sigma, \tau : \texttt{Stream}^*).\ (\sigma \equiv \tau) \implies (\texttt{hd}^*\ \sigma = \texttt{hd}^*\ \tau) \\ &\Pi(\sigma, \tau : \texttt{Stream}^*).\ (\sigma \equiv \tau) \implies \texttt{cls}\ (\texttt{tl}^*\ \sigma) = \texttt{cls}\ (\texttt{tl}^*\ \tau)\end{aligned}$$

Proof. Suppose that we have $(\sigma, \tau : \texttt{Stream}^*)$ such that $(\sigma \equiv \tau)$. This means that we have $\sigma = \texttt{corec}_s^*\ X\ h\ t\ x$ and $\tau = \texttt{corec}_s^*\ Y\ h'\ t'\ (f\ x)$ for suitably typed X, Y, h, h', t, t', f, and x such that $h \circ f = h'$ and $t' \circ f = f \circ t$. Note that by assumption we have $h\,x = h'(f\,x)$, and we conclude that $\texttt{hd}^*\ \sigma = h\,x = h'(f\,x) = \texttt{hd}^*\ \tau$.

Next, we show that $\texttt{cls}\ (\texttt{tl}^*\ \sigma) = \texttt{cls}\ (\texttt{tl}^*\ \tau)$, and it suffices to prove $\texttt{tl}^*\ \sigma \equiv \texttt{tl}^*\ \tau$. Note that we have $\texttt{tl}^*\ \tau = \texttt{corec}_s^*\ Y\ h'\ t'\ (t' \circ (f\ x))$ and $\texttt{tl}^*\sigma = \texttt{corec}_s^*\ X\ h\ t\ (t\ x)$. By assumption, we also have $\texttt{corec}_s^*\ Y\ h'\ t'\ (t' \circ (f\ x)) = \texttt{corec}_s^*\ Y\ h'\ t'\ (f \circ (t\ x))$. The desired statement follows because $\texttt{corec}_s^*\ Y\ h'\ t'\ (f \circ (t\ x)) \equiv \texttt{corec}_s^*\ X\ h\ t\ (t\ x)$. ◀

We define the new head and tail functions `hd` and `tl` and the corecursor as lifted functions.

▶ **Definition 4.9.** We define the destructor $\texttt{hd} : \texttt{Stream} \to E$ to be $\overline{\texttt{hd}^*} = \texttt{rec}_\texttt{q}\ E\ \texttt{hd}^*\ \texttt{EqHd}$, and the destructor $\texttt{tl} : \texttt{Stream} \to \texttt{Stream}$ to be $\overline{\texttt{cls} \circ \texttt{tl}^*} = \texttt{rec}_\texttt{q}\ \texttt{Stream}\ (\texttt{cls} \circ \texttt{tl}^*)\ \texttt{EqTl}$. Finally, we define the corecursor $\texttt{corec}_\texttt{s} : \Pi(X : \mathcal{U}).(X \to E) \to (X \to X)$ to be $\texttt{cls} \circ \texttt{corec}_\texttt{s}^*$.

In the remainder of this section, we show that `Stream` is the final coalgebra. We first remark that this stream type still satisfies the β-rules. More specifically, we have $\texttt{hd}\ (\texttt{corec}_\texttt{s}\ X\ h\ t\ x) \overset{\beta}{=} h\ x$ and $\texttt{tl}\ (\texttt{corec}_\texttt{s}\ X\ h\ t\ x) \overset{\beta}{=} \texttt{corec}_\texttt{s}\ X\ h\ t\ (t\ x)$.

Next, we have that $(\texttt{corec}_\texttt{s}^*\ X\ h\ t)$ and `cls` are $\mathcal{S}$-morphisms, which again follows from the β-rules of $\texttt{Stream}^*$ and `quot`.

▶ **Lemma 4.10.** *The class function* $(\texttt{cls} : \texttt{Stream}^* \to \texttt{Stream})$ *is an* $\mathcal{S}$*-morphism. In addition, for all* $(X : \mathcal{U})$, $(h : X \to E)$, $(t : X \to X)$ *and* $(x : X)$, *we have that* $(\texttt{corec}_s^*\ X\ h\ t)$ *forms an* $\mathcal{S}$*-morphism* $(X \to \texttt{Stream}^*)$.

Before we show that the η-rule holds for `Stream`, we first prove the following lemmas.

▶ **Lemma 4.11.** *Given* $(\sigma : \texttt{Stream}^*)$, *we have*

$$(\texttt{corec}_s^*\ \texttt{Stream}^*\ \texttt{hd}^*\ \texttt{tl}^*\ \sigma) \equiv (\texttt{corec}_s^*\ \texttt{Stream}\ \texttt{hd}\ \texttt{tl}\ (\texttt{cls}\ \sigma)).$$

Proof. By Lemma 4.10, we have that `cls` is a morphism. Therefore, the `CoLimStr` predicate, instantiated using the obvious choices for X, Y, h, h', t, t', f and x follows directly. ◀

▶ **Lemma 4.12.** *Given* $(\sigma : \texttt{Stream}^*)$, *we have* $(\texttt{corec}_s^*\ \texttt{Stream}^*\ \texttt{hd}^*\ \texttt{tl}^*\ \sigma) \equiv \sigma$.

Proof. Following the definition of `CoLimStr` in Definition 4.5, we need to find some $X : \mathcal{U}$, $x : X$, $h : X \to X$, $t : X \to X$ and a stream morphism $f : X \to \texttt{Stream}^*$ such that $\sigma = \texttt{corec}_\texttt{s}^*\ X\ h\ t\ x$ and $\sigma = f\ x$. We have $\sigma : \texttt{Stream}^*$, where $\texttt{Stream}^* = \exists(X : \mathcal{U}).X \times (X \to E) \times (X \to X)$. We destruct σ to get $X : \mathcal{U}$, $x : X$, $h : X \to X$, $t : X \to X$ with $\sigma = \texttt{pack}\ X\ \langle x, h, t\rangle = \texttt{corec}_\texttt{s}^*\ X\ h\ t\ x$. For the other requirement, we take $f := \texttt{corec}_\texttt{s}^*\ X\ h\ t$, which is indeed a morphism by Lemma 4.10. ◀

▶ **Lemma 4.13.** *We have* $\texttt{corec}_s\ \texttt{Stream}\ \texttt{hd}\ \texttt{tl} = \texttt{id}_{\texttt{Stream}}$.

Proof. See the Appendix (Lemma B.1). ◀

We are now ready to show that the recursor is unique.

▶ **Theorem 4.14.** *For all* $(X : \mathcal{U}), (h : X \to E), (t : X \to X)$ *and* $(f : X \to \texttt{Stream})$, *if* $(\texttt{MorphStream}\ X\ h\ t\ \texttt{Stream}\ \texttt{hd}\ \texttt{tl}\ f)$, *then we have* $(f = \texttt{corec}_s\ X\ h\ t)$

Proof. By function extensionality, it is enough to prove $(f\ x = \texttt{corec}_\texttt{s}\ X\ h\ t\ x)$ for all $(x : X)$. We apply Lemma 4.13 and the fact that $(\texttt{corec}_\texttt{s}^*\ X\ h\ t\ x)$ and $(\texttt{corec}_\texttt{s}^*\ \texttt{Stream}\ \texttt{hd}\ \texttt{tl}\ (f\ x))$ are related by `CoLimStr`: $f\ x \overset{4.13}{=} \texttt{corec}_\texttt{s}^*\ \texttt{Stream}\ \texttt{hd}\ \texttt{tl}\ (f\ x) = \texttt{corec}_\texttt{s}^*\ X\ h\ t\ x$. ◀

4.3 Stream bisimulation principle

We now prove the *coinduction principle*, which states that two streams are equal if they are *bisimilar*, that is: they have the same observable behavior. Formally, two streams are bisimilar if there is a bisimulation relation that relates them. We define the notion of R being a bisimulation relation, and then we define the relation `BiSim` that relates two streams if they can be related via some bisimulation R.

▶ **Definition 4.15.** We say that R is a **bisimilation** if for all $\sigma, \tau : \texttt{Stream}$ we have that $\texttt{hd}\ \sigma = \texttt{hd}\ \tau$ and $R\ (\texttt{tl}\ \sigma)\ (\texttt{tl}\ \tau)$ whenever $R\ \sigma\ \tau$. The type expressing that R is a bisimulation is denoted by $\texttt{IsBiSim}\ R$. We say that σ and τ are **bisimilar** if there a bisimulation R such that $R\ \sigma\ \tau$. We write $(\sigma \sim \tau)$ or $\texttt{BiSim}\ \sigma\ \tau$ to denote that σ and τ are bisimilar.

▶ **Theorem 4.16.** *We have the coinduction proof principle, which means that for all $\sigma, \tau :$ $\texttt{Stream}$, we have $\sigma \sim \tau$ if and only if $\sigma = \tau$.*

Proof. ($\Longleftarrow$) follows because $=$ is a bisimulation. For ($\Longrightarrow$), assume we have $\sigma \sim \tau$. Consider the following quotient $\texttt{Stream}/\sim := \texttt{quot Stream BiSim}$. We define a function $f : \texttt{Stream}/\sim \to \texttt{Stream}$ such that $f \circ \texttt{cls}_\sim : \texttt{Stream} \to \texttt{Stream}$ is a $\mathcal{S}$-morphism. (See the Appendix, Theorem 2 for details.) By the η-rule for streams (Theorem 4.14), we find that $f = \texttt{id}_{\texttt{Stream}}$. Therefore $\sigma = f(\texttt{cls}_\sim\ \sigma) = f(\texttt{cls}_\sim\ \tau) = \tau$. ◀

5 Ensuring the η-rules in general

We used the limit predicate to encode the inductive data types, similar to [4]. For coinductive data types, we dually used the colimit. It is also possible to use other predicates. For example, one can directly encode the induction principle within the embedding, by creating a predicate `Ind` as was done in [31]. It is also possible to encode the η-rule directly by representing the uniqueness requirement, using a predicate `Unq`. More precisely, in terms of the quotient type example: we define `quot` from $\texttt{quot}^*$ as a subtype

$$\texttt{quot}\ (D : \mathcal{U})\ (R : D \to D \to \mathcal{U}) := \Sigma(q : \texttt{quot}^*\ D\ R).\texttt{QuotPred}\ q$$

where `QuotPred` expresses the η-rule. We have different choices for `QuotPred`

$$\begin{aligned}
\texttt{LimQuot}\ (q : \texttt{quot}^*\ D\ R) &:= \Pi(X, Y : \mathcal{U}).\Pi\tbinom{g : D \to X}{g' : D \to Y}.\Pi(f : X \to Y).\\
&\quad \Pi(H : (\texttt{EqCls}\ g\ R))(H' : (\texttt{EqCls}\ g'\ R)).\\
&\quad f \circ g = g' \implies f\ (\texttt{rec}^*_{\texttt{q}}\ X\ g\ H\ q) = \texttt{rec}^*_{\texttt{q}}\ Y\ g'\ H'\ q\\
\texttt{UnqQuot}\ (q : \texttt{quot}^*\ D\ R) &:= \Pi(X : \mathcal{U}).\Pi(g : D \to X).\Pi(f : \texttt{quot}^*\ D\ R \to X).\\
&\quad \Pi(H : (\texttt{EqCls}\ g\ R)).\\
&\quad (\Pi(d : D).f(\texttt{cls}^*\ d) = g\ d) \implies f\ q = \texttt{rec}^*_{\texttt{q}}\ X\ g\ H\ q\\
\texttt{IndQuot}\ (q : \texttt{quot}^*\ D\ R) &:= \Pi(P : \texttt{quot}^*\ D\ R \to \mathcal{U}).\\
&\quad (\Pi(d : D).P(\texttt{cls}^*\ d)) \implies P\ q
\end{aligned}$$

For `LimQuot` and `UnqQuot`, the types we obtain are equivalent, because `LimQuot` and `UnqQuot` are proof-irrelevant. For `IndQuot` this is only the case if we define `IndQuot` by quantifying only over `hProp` valued predicates. A similar analysis can be made for the `Stream` data type. In the rest of the paper (and the Appendix) these predicates will appear for W-types and M-types. The table below summarizes the different encodings and where they can be found for W-types and M-types in this paper.

`LimW`	Definition 6.2	`CoLimM`	Definition 7.2
`UnqW`	Theorem 6.6	`UnqM`	Theorem 7.8
`IndW`	Theorem 6.7	`CoIndM`	Theorem 7.10

6 W-types

We introduce W-types, which generalize inductive types such as natural numbers, lists, and trees. They were introduced by Martin-Löf [22, Wellorderings] as a way to define inductive data structures within type theory. We give the well-known impredicative definition of W-types. This impredicative definition yields the expected β-rule. We then define an impredicative encoding that ensures the η-rule also holds. Using this η-rule one can prove the induction principle for W-types. In the literature, e.g. [35, 5.3 W-Types], W-types are defined as the initial algebra of the functor $\mathcal{W}$ given by $\mathcal{W}(X) := \Sigma(a : A).B(a) \to X$.

▶ **Definition 6.1.** We define the **impredicative W-type** $W^*(a : A).B(a)$ as the type $\Pi(X : \mathcal{U}).(\Pi(a : A).(B(a) \to X) \to X) \to X$. For $a : A$ and $r : B(a) \to W^*_{AB}$, we define

$$\mathtt{sup}^*\ a\ r := \lambda(X : \mathcal{U}).\lambda(g : \Pi(a : A).(B(a) \to X) \to X).g\ a\ (\lambda b.r\ b\ X\ g).$$

Finally, given $X : \mathcal{U}$ and $g : \Pi(a : A).(B(a) \to X) \to X$, we define $\mathtt{rec}^*_{\mathtt{W}}\ X\ g$ to be $\lambda(w : W^*_{AB}).w\ X\ g$.

Note that $\mathtt{rec}^*_{\mathtt{W}}\ X\ g\ (\mathtt{sup}^*\ a\ r) \overset{\beta}{=} g\ a\ ((\mathtt{rec}^*_{\mathtt{W}}\ X\ g) \circ r)$. Next, we study the categorical definition of W-types as the initial algebra of the $\mathcal{W}$-functor. We first define $\mathcal{W}$-morphisms.

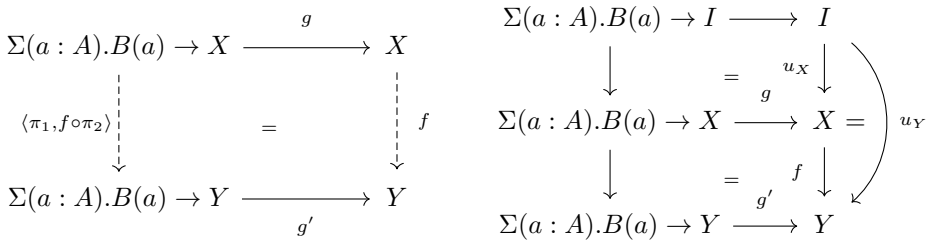


Figure 3 Commutative diagram of $\mathcal{W}$-morphisms (left); Uniqueness requirement of the initial $\mathcal{W}$-algebra (right).

Suppose that we have $(X, Y : \mathcal{U})$, $(g : \Pi(a : A).(B(a) \to X) \to X)$ and $(g' : \Pi(a : A).(B(a) \to Y) \to Y)$. A map $(f : X \to Y)$ is a **$\mathcal{W}$-morphism** if we have $f(g(\langle a, t\rangle) = g'(\langle a, f(t)\rangle)\ (\forall(a : A), (t : B(a) \to X))$, which is illustrated in Figure 3 (left). We write $\mathtt{MorphW}\ f$ for the type that states that f is a $\mathcal{W}$-morphism.

We write out the uniqueness requirement of the initial $\mathcal{W}$-algebra. This η-rule states that for each $\mathcal{W}$-morphism $(f : X \to Y)$, we have $f \circ u_X = u_Y$. We encode this requirement within the definition of an inductive W-type, which is an embedding of the previously defined impredicative definition Definition 6.1. So we have $W(a : A).B(a) \hookrightarrow W^*(a : A).B(a)$. We use a Σ-type to encode a proof term of the uniqueness requirement of Figure 3 (right).

▶ **Definition 6.2** (`LimW`)**.** An element $w : W^*(a : A).B(a)$ satisfies the predicate `LimW` if for all types $X, Y : \mathcal{U}$ and functions $g : (\Pi(a : A).B(a) \to X) \to X$ and $g' : (\Pi(a : A).B(a) \to Y) \to Y$ we have $f\ (\mathtt{rec}^*_{\mathtt{W}}\ X\ g\ w) = \mathtt{rec}^*_{\mathtt{W}}\ Y\ g'\ w$ for all W-morphisms f. The **inductive W-type** $W(a : A).B(a)$ is defined to be $\Sigma(w : W^*(a : A).B(a)).\mathtt{LimW}\ w$. Finally, for all $(X : \mathcal{U})$ and $(g : \Pi(a : A).(B(a) \to X) \to X)$ we define its recursor $\mathtt{rec}_{\mathtt{W}}\ X\ g$ to be $\lambda(w : W(a : A).B(a)).\mathtt{rec}^*_{\mathtt{W}}\ X\ g\ (\mathtt{pr1}\ w)$.

We have to show that elements of the original W-type satisfy the `LimW` predicate.

▶ **Lemma 6.3** (`LimWSup`)**.** *We have* $\mathtt{LimW}\ (\mathtt{sup}^*\ a\ (\mathtt{pr1} \circ r))$ *for* $(a : A)$ *and* $(r : B(a) \to W_{AB})$.

Proof. Let $(X, Y : \mathcal{U})$, $(g : \Pi(a : A).(B(a) \to X) \to X)$, $(g' : \Pi(a : A).(B(a) \to Y) \to Y)$ and $(f : X \to Y)$ be given with $(\mathtt{MorphW}\ X\ g\ Y\ g'\ f)$. We need to show $f\ (\mathtt{rec}^*_{\mathtt{W}}\ X\ g\ (\mathtt{sup}^*\ a\ (\mathtt{pr1} \circ r))) = \mathtt{rec}^*_{\mathtt{W}}\ Y\ g'\ (\mathtt{sup}^*\ a\ (\mathtt{pr1} \circ r))$. Note that by instantiating $(\mathtt{MorphW}\ X\ g\ Y\ g'\ f)$ with a and $(\mathtt{rec}^*_{\mathtt{W}}\ X\ g) \circ \mathtt{pr1} \circ r$, we obtain the following equation.

$$f\ (g\ a\ ((\mathtt{rec}^*_{\mathtt{W}}\ X\ g) \circ \mathtt{pr1} \circ r)) = g'\ a\ (f \circ (\mathtt{rec}^*_{\mathtt{W}}\ X\ g) \circ \mathtt{pr1} \circ r) \tag{1}$$

For all $(b : B(a))$ we have $(r\ b) : W_{AB}$ and so $(\mathtt{pr2}\ (r\ b) : (\mathtt{LimW}\ (\mathtt{pr1}\ (r\ b)))$. From this we derive the following equation.

$$f \circ (\mathtt{rec}^*_{\mathtt{W}}\ X\ g) \circ (\mathtt{pr1}\ \circ r) = (\mathtt{rec}^*_{\mathtt{W}}\ Y\ g') \circ (\mathtt{pr1}\ \circ r) \tag{2}$$

We conclude as follows.

$$\begin{array}{rcll} f\ (\mathtt{rec}^*_{\mathtt{W}}\ X\ g\ (\mathtt{sup}^*\ a\ (\mathtt{pr1}\ \circ r))) & \overset{\beta}{=} & f\ (g\ a\ ((\mathtt{rec}^*_{\mathtt{W}}\ X\ g) \circ (\mathtt{pr1}\ \circ r))) & \overset{(1)}{=} \\ g'\ a\ (f \circ (\mathtt{rec}^*_{\mathtt{W}}\ X\ g)) \circ (\mathtt{pr1}\ \circ r)) & \overset{(2)}{=} & g'\ a\ (\mathtt{rec}^*_{\mathtt{W}}\ Y\ g') \circ (\mathtt{pr1}\ \circ r)) & \\ & = & \mathtt{rec}^*_{\mathtt{W}}\ Y\ g'\ (\mathtt{sup}^*\ a\ (\mathtt{pr1}\ \circ r)) & \end{array}$$

◀

▶ **Definition 6.4.** For all $(a : A)$ and $(r : B(a) \to W_{AB})$ we define the constructor $\mathtt{sup}\ a\ r$ to be $\langle \mathtt{sup}^*\ a\ (\mathtt{pr1}\ \circ r),\ \mathtt{LimWSup}\ a\ (\mathtt{pr1}\ \circ r)\rangle$.

Note that the β-rule holds for $W(a : A).B(a)$, meaning that we have $\mathtt{rec}_{\mathtt{W}}\ X\ g\ (\mathtt{sup}\ a\ r) \overset{\beta}{=} g\ a\ ((\mathtt{rec}_{\mathtt{W}}\ X\ g) \circ r)$. Next, we look at the η-rule, which says that the morphism $(\mathtt{rec}_{\mathtt{W}}\ X\ g)$ is unique. To prove the complete η-rule, we first show a special case.

▶ **Lemma 6.5** (RecWId). *We have* $\mathtt{rec}_{\mathtt{W}}\ W_{AB}\ \mathtt{sup} = \mathtt{id}_{W_{AB}}$.

Proof. Using function extensionality and Lemma 2.1, it suffices to prove that $(\mathtt{pr1}\ w)\ X\ g$ is equal to $(\mathtt{pr1}\ (\mathtt{rec}_{\mathtt{W}}\ W_{AB}\ \mathtt{sup}\ w))\ X\ g$ for all $X : \mathcal{U}$, $g : \Pi(a : A).(B(a) \to X) \to X)$ and $w : W_{AB}$. Since $(\mathtt{pr2}\ w)$ is of type $\mathtt{LimW}\,(\mathtt{pr1}\ w)$, we have $(\mathtt{rec}_{\mathtt{W}}\ X\ g)\ (\mathtt{rec}^*_{\mathtt{W}}\ W_{AB}\ \mathtt{sup}\ (\mathtt{pr1}\ w)) = (\mathtt{rec}^*_{\mathtt{W}}\ X\ g\ (\mathtt{pr1}\ w))$. The desired statement now follows from the definitions of $\mathtt{rec}_{\mathtt{W}}$ and $\mathtt{rec}^*_{\mathtt{W}}$ and the fact that $\mathtt{rec}_{\mathtt{W}}\ X\ g$ is a $\mathcal{W}$-morphism. ◀

We now show the η-rule for the W-type, stating that the recursors $(\mathtt{rec}_{\mathtt{W}}\ X\ g)$ are unique.

▶ **Theorem 6.6** (UnqW). *Given* $(X : \mathcal{U})$, $(g : \Pi(a : A).(B(a) \to X) \to X)$ *and* $(f : W_{AB} \to X)$, *we have* $f = \mathtt{rec}_{\mathtt{W}}\ X\ g$ *whenever* $(\mathtt{MorphW}\ W_{AB}\ \mathtt{sup}\ X\ g\ f)$.

Proof. Let f be a $\mathcal{W}$-morphism and let $(w : W_{AB})$. We have to show that $f\ w = \mathtt{rec}_{\mathtt{W}}\ X\ g\ w$. By Theorem 6.6, it suffices to show that $f\ (\mathtt{rec}_{\mathtt{W}}\ W_{AB}\ \mathtt{sup}\ w) = \mathtt{rec}^*_{\mathtt{W}}\ X\ g\ (\mathtt{pr1}\ w)$. Since $(p := \mathtt{pr2}\ w)$ has type $\mathtt{LimW}\,(\mathtt{pr1}\ w)$, we have that $f\ (\mathtt{rec}^*_{\mathtt{W}}\ W_{AB}\ \mathtt{sup}\ (\mathtt{pr1}\ w))$ and $\mathtt{rec}^*_{\mathtt{W}}\ X\ g\ (\mathtt{pr1}\ w)$ are equal. The desired statement follows from the following equalities.

$$f\ (\mathtt{rec}_{\mathtt{W}}\ W_{AB}\ \mathtt{sup}\ w) = f\ (\mathtt{rec}^*_{\mathtt{W}}\ W_{AB}\ \mathtt{sup}\ (\mathtt{pr1}\ w)) = \mathtt{rec}^*_{\mathtt{W}}\ X\ g\ (\mathtt{pr1}\ w).$$

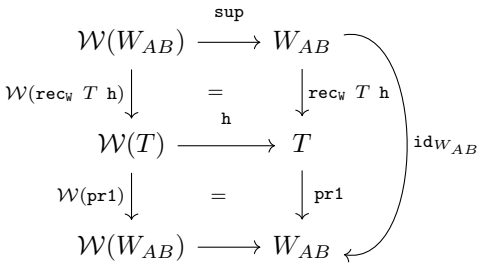
◀

We now show that we have an induction principle for our impredicative encoding W_{AB}. We follow the same approach as in the proof for quotients which has also been used in [5] and already appears in [10]. Let P be a predicate on W_{AB}. For $(a : A)$, and $(r : (B\ a) \to W_{AB})$, we write $\mathtt{Hyp}_W\ a\ r$ for the type $\Pi(h : \Pi(b : B\ a).P\ (r\ b)) \implies P\ (\mathtt{sup}\ a\ r)$. The **induction principle of W-types** says that we have $P\ w$ for each $(w : W_{AB})$ whenever we have $\mathtt{Hyp}_W\ a\ r$ for all $(a : A)$ and $(r : (B\ a) \to W_{AB})$. To prove this induction principle holds for W_{AB}, we use Theorem 6.6.

▶ **Theorem 6.7** (IndW). *The W_{AB} satisfies the induction principle for W-types.*

Proof. Let $(P : W_{AB} \to \mathcal{U})$. We assume that we have a term $(H\ a\ r : (\mathtt{Hyp}_W\ a\ r))$ for all $(a : A)$ and $(r : (B\ a) \to W_{AB})$. The proof proceeds as follows.

1. Using H we show that $(T := \Sigma(w : W_{AB}).P\ w)$ together with some function $\mathtt{h}$ forms a $\mathcal{W}$-algebra.
2. We have that $\mathtt{pr1}$ is a $\mathcal{W}$-morphism $(\mathtt{pr1} : T \to W_{AB})$.
3. We have the unique $\mathcal{W}$-morphism $(\mathtt{rec_W}\ T\ \mathtt{h} : W_{AB} \to T)$.
4. We have that $(\mathtt{pr1} \circ (\mathtt{rec_W}\ T\ \mathtt{h}))$ is a $\mathcal{W}$-morphsim $(\mathtt{pr1} \circ (\mathtt{rec_W}\ T\ \mathtt{h}) : W_{AB} \to W_{AB})$.
5. We have that $\mathtt{id}_{W_{AB}}$ is also an $\mathcal{W}$-morphism from W_{AB} to W_{AB}.
6. By the η-rule we conclude that that $\mathtt{pr1} \circ (\mathtt{rec_W}\ T\ \mathtt{h}) = \mathtt{id}_{W_{AB}}$.
7. For all $(w : W_{AB})$ we have that $\mathtt{pr1}\ (\mathtt{rec_W}\ T\ \mathtt{h}\ w) = w$ and thus conclude that $(\mathtt{pr2}\ (\mathtt{rec_W}\ T\ \mathtt{h}\ w) : P\ w)$.



◀

7 M-types

Finally, we look at impredicative encodings of M-types [36], which are the dual notion of W-types. They generalize coinductive types such as streams, infinite lists, and non-wellfounded trees. We follow the approach of Section 4: we first give an impredicative definition, which only satisfies the desired β-rule. Then we take a quotient to ensure the η-rule. We finish this section by proving the coinduction principle for M-types.

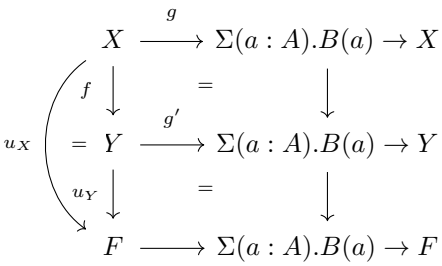
M-types are described by specifying the destructors. More specifically, we look at the final coalgebra of the functor $\mathcal{M}$ given by $\Sigma(a : A).B(a) \to X$. Note that the functor $\mathcal{M}$ is defined in the same way as $\mathcal{W}$ in Section 6 since both represent polynomial endofunctors. However, we use a different name that is more suggestive in the context of M-types. The impredicative encoding for M-types is defined using existential types.

▶ **Definition 7.1.** We define the **impredicative M-type** $M^*(a : A).B(a)$ to be $\exists(X : \mathcal{U}).X \times (X \to \Sigma(a : A).B(a) \to X)$. For $(X : \mathcal{U}), (r : X \to \Sigma(a : A).B(a) \to X)$, and $x : X$, we define **corecursor** $\mathtt{corec}^*_{\mathtt{M}}\ X\ r\ x$ to be $\mathtt{pack}\ X\ \langle x, r\rangle$. Finally, we define the coalgebra map $\mathtt{elimM}^* : M^*_{AB} \to \Sigma(a : A).B(a) \to M^*_{AB}$ to be

$$\mathtt{rec}_\exists\ (\Sigma(a : A).B(a) \to M^*_{AB})(\lambda X.\lambda\langle x, f\rangle.\ \langle \mathtt{pr1}\ (f\ x),\ \lambda b.\mathtt{pack}\ X\ \langle \mathtt{pr2}\ (f\ x)\ b, f\rangle\ \rangle\ ).$$

The function $\mathtt{elimM}^*$ takes an $(m : M^*_{AB})$ and produces a value of type $(\Sigma(a : A).B(a) \to M^*_{AB})$. It does so using the recursor of the existential type to obtain a type $(X : \mathcal{U})$, an element $(x : X)$, and a function $(X \to \Sigma(a : A).B(a) \to X)$. This allows us to construct the desired element. Note that the β-rule holds for M^*_{AB}, since we have $\mathtt{elimM}^*\ (\mathtt{corec}^*_{\mathtt{M}}\ X\ r\ x) \overset{\beta}{=} \langle \mathtt{pr1}\ (r\ x),\ (\mathtt{corec}^*_{\mathtt{M}}\ X\ r) \circ (\mathtt{pr2}\ (r\ x))\rangle$.

The next step is to refine the type M^*_{AB} to obtain the final coalgebra for $\mathcal{M}$. We first define $\mathcal{M}$-morphisms. Suppose that we have types X and Y with functions $g : X \to \Sigma(a : A).B(a) \to X$ and $g' : Y \to \Sigma(a : A).B(a) \to Y$. A function $(f : X \to Y)$ is an $\mathcal{M}$-morphism if $g' \circ f = \langle \pi_1, f \circ \pi_2 \rangle \circ g$. We write $\texttt{MorphM}\ f$ for the type that expresses that f is an $\mathcal{M}$-morphism. The uniqueness requirement for a final $\mathcal{M}$-coalgebra F states that for each $\mathcal{M}$-morphism $(f : X \to Y)$, we have $u_X = f \circ u_Y$ as indicated in the diagram below.



As we saw with $\texttt{Stream}^*$, we define a colimit relation $\texttt{CoLimM}$ that relates two M-types if they are related by an $\mathcal{M}$-morphism. We then create a quotient using this relation such that two M-types that are related by $\texttt{CoLimM}$ are equated. We finally prove that this quotient satisfies the η-rule.

▶ **Definition 7.2** ($\texttt{CoLimM}$)**.** Let $(m, n : M_{AB})$. The predicate $(\texttt{CoLimM}\ m\ n)$ holds if there are coalgebras (X, g) and (Y, g'), a $\mathcal{M}$-morphism f from (X, g) to (Y, g'), and $x : X$ such that $m = \texttt{corec}^*_{\texttt{M}}\ X\ g\ x$ and $n = \texttt{corec}^*_{\texttt{M}}\ Y\ g'\ (f\ x)$. We write $(m \equiv n)$ to denote $(\texttt{CoLimM}\ m\ n)$.

▶ **Definition 7.3.** We define the **coinductive M-type** to be $\texttt{quot}\ (M^*(a : A).B(a))\ \texttt{CoLimM}$, and we denote this type by M_{AB}. For $(X : \mathcal{U})$, $(r : X \to \Sigma(a : A).B(a) \to X)$, and $x : X$, we define $\texttt{corec}_{\texttt{M}}\ X\ r\ x$ to be $\texttt{cls}\ (\texttt{corec}^*_{\texttt{M}}\ X\ r\ x)$.

In order to lift the $\texttt{elimM}^*$ function, we need to show that it respects the relation $\texttt{CoLimM}$.

▶ **Lemma 7.4.** *If $m \equiv n$, then we have $(\texttt{pr1}\ (\texttt{elimM}^*\ m)) = (\texttt{pr1}\ (\texttt{elimM}^*\ n))$ and $(\texttt{cls}\ (\texttt{pr2}\ (\texttt{elimM}^*\ m))) = (\texttt{cls}\ (\texttt{pr2}\ (\texttt{elimM}^*\ n)))$.*

The first statement in Lemma 7.4 follows from unfolding the definitions, and the other is proven in Appendix (Lemma B.3). Now we lift $\texttt{elimM}^*$.

▶ **Definition 7.5.** We define $\texttt{elimM} : \Pi(m : M_{AB}) \to M_{AB} \to \Sigma(a : A).B(a) \to M_{AB}$ to be $\overline{\langle \texttt{pr1}, \texttt{cls} \circ \texttt{pr2} \rangle} \circ \texttt{elimM}^*$.

This improved M-type satisfies the expected computation rule, which follows by unfolding the definitions and doing the computations. More specifically, we have:

$$\texttt{elimM}\ (\texttt{corec}_{\texttt{M}}\ X\ r\ x) \overset{\beta}{=} \langle \texttt{pr1}\ (r\ x),\ \texttt{corec}_{\texttt{M}}\ X\ r\ (\texttt{pr2}\ (r\ x)) \rangle$$

Next, we prove that the η-rule holds. We first note that $(\texttt{corec}^*_{\texttt{M}}\ X\ g)$ and $\texttt{cls}$ are $\mathcal{M}$-morphisms. This follows by unfolding the definitions. As a consequence, we get that $(\texttt{corec}^*_{\texttt{M}}\ M^*_{AB}\ \texttt{elimM}^*\ (\texttt{cls}\ m)) \equiv (\texttt{corec}^*_{\texttt{M}}\ M_{AB}\ \texttt{elimM}\ m)$ for each $m : M^*_{AB}$.

We need two lemmas, which are proven in Appendix (Lemma B.4 and Lemma B.5).

▶ **Lemma 7.6.** *We have $m \equiv (\texttt{corec}^*_{M}\ M^*_{AB}\ \texttt{elimM}^*\ m)$ for each $(m : M^*_{AB})$.*

▶ **Lemma 7.7.** *We have $\texttt{corec}_{M}\ M_{AB}\ \texttt{elimM} = \texttt{id}_{M_{AB}}$.*

▶ **Theorem 7.8** (UnqM). *For all* $(X : \mathcal{U})$, $(g : X \to \Sigma(a : A).B(a) \to X)$ *and* $(f : X \to M_{AB})$ *we have* $(f = \mathtt{corec}_M\ X\ g)$ *whenever* $(\mathit{MorphM}\ X\ g\ M_{AB}\ \mathit{elimM}\ \ f)$.

Proof. We use function extensionality by fixing some $(x : X)$. Note that $(\mathtt{corec_M}\ X\ g\ x)$ and $(\mathtt{corec_M}\ M_{AB}\ \mathtt{elimM}\ (f\ x))$ are related by CoLimM, so we get

$$f\ x = \mathtt{corec_M}\ M_{AB}\ \mathtt{elimM}\ (f\ x) = \mathtt{corec_M}\ X\ g\ x. \qquad \blacktriangleleft$$

We now show that M_{AB} satisfies the coinduction principle. This principle states that two inhabitants of M_{AB} are equal if they are bisimilar. We start by defining bisimilarity.

▶ **Definition 7.9.** A relation $R : M_{AB} \to M_{AB} \to \mathcal{U}$ is a **bisimulation** if for all $(m, n : M_{AB})$ such that $R\ m\ n$, we have $\mathtt{pr1}\ (\mathtt{elimM}\ m) = \mathtt{pr1}\ (\mathtt{elimM}\ n)$ and if for each $b : B(\mathtt{pr1}\ (\mathtt{elimM}\ m))$, we have $R\ (\mathtt{pr2}\ (\mathtt{elimM}\ m)\ b)\ (\mathtt{pr2}\ (\mathtt{elimM}\ n)\ b)$. We say that $m, n : M_{AB}$ are bisimilar if there is a bisimulation R such that $R\ m\ n$. We write $\mathtt{IsBisimM}\ R$ for the type expressing that R is a bisimulation, and we write $\mathtt{BiSimM}\ m\ n$ or $m \sim n$ to indicate that m and n are bisimilar.

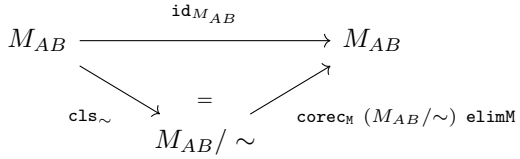
Note that propositional equality on M-types is a bisimulation. We finish this section with coinduction for M-types.

▶ **Theorem 7.10** (CoIndM). *For al* $(m, n : M_{AB})$ *we have* $m \sim n$ *if and only if* $m = n$.

Proof. ($\Longleftarrow$) This follows because equality is a bisimulation.
($\Longrightarrow$) Let $(m, n : M_{AB})$ be M-types and assume $(m \sim n)$. We define the following quotient.

$$M_{AB}/\sim := \mathtt{quot}\ M_{AB}\ \mathtt{BiSimM}$$

By $(m \sim n)$ there exists an bisimulation $(R : M_{AB} \to M_{AB} \to \mathcal{U})$. Let R be that bisimulation. We lift the M-destructor and define $\mathtt{elimX} := \overline{\langle \mathtt{pr1}, \mathtt{cls}_\sim \circ \mathtt{pr2} \rangle \circ \mathtt{elimM}}$. This construction is well-defined by the definition of IsBisimM and a proof is almost identical to the (lengthy though not complicated) proof of Lemma 7.4. We show that the following diagram commutes:



We define $f := (\mathtt{corec_M}\ (M_{AB}/\sim)\ \mathtt{elimX}) \circ \mathtt{cls}_\sim$. We have that $(f : M_{AB} \to M_{AB})$ forms an $\mathcal{M}$-morphism, which follows from these equations, for $(x : X)$:

$$\mathtt{pr1}\ (\mathtt{elimM}\ (f\ x)) = \mathtt{pr1}\ (\mathtt{elimM}\ x) \tag{3}$$

$$\mathtt{pr2}\ (\mathtt{elimM}\ (f\ x)) = f \circ (\mathtt{pr2}\ (\mathtt{elimM}\ x)) \tag{4}$$

Both equations can be shown using the β-rule for quotients and the β-rule for M-types. From the η-rule of M-types (Theorem 7.8), we derive that

$$f := (\mathtt{corec_M}\ M_{AB}\ \mathtt{elimX}) \circ \mathtt{cls}_\sim = (\mathtt{corec_M}\ M_{AB}\ \mathtt{elimM}) \tag{5}$$

Because $(m \sim n)$ we also have

$$\mathtt{cls}_\sim\ m = \mathtt{cls}_\sim n \tag{6}$$

Now we derive the following to conclude $m = n$.

$$\begin{aligned} m \overset{7.7}{=}\ & \mathtt{corec_M}\ M_{AB}\ \mathtt{elimM}\ m \overset{5}{=} \mathtt{corec_M}\ (M_{AB}/\sim)\ \mathtt{elimX}\ (\mathtt{cls}_\sim\ m) \\ \overset{6}{=}\ & \mathtt{corec_M}\ (M_{AB}/\sim)\ \mathtt{elimX}\ (\mathtt{cls}_\sim\ n) \overset{5}{=} \mathtt{corec_M}\ M_{AB}\ \mathtt{elimM}\ n \overset{7.7}{=} n \end{aligned} \qquad \blacktriangleleft$$

8 Conclusion and Future work

We have extended the technique of [4] of encoding η-rules within the impredicative definition of data types to quotients, streams, W-types, and M-types. For quotients and W-types, we distilled the uniqueness requirement from initial algebra semantics and encoded it using Σ-types. We showed that these improved types satisfy the η-rules and proved the induction principles. Using the inductive quotient type and a refined existential type, we dualized this technique to coinductive data-types and we defined streams and general M-types. We showed that these types satisfy the η-rule and that bisimilarity coincides with equality.

There are various future topics to study. First, this work can be formalized in a proof assistant such as Rocq [34] or Lean [11]. Initial steps have been made to formalize parts of the paper [4] by Awodey `https://github.com/awodey/Impredicative`. Furthermore, we have not looked at higher inductive types. In this case, one should drop the `UIP` axiom and use a notion of `Set`. Also, it is known that the definable data types in System F do not always produce functions that compute properly, because functions are defined by *iteration*, and not by *primitive recursion*, which makes the destructor functions for inductive data types (e.g. predecessor for the natural numbers) hard to define and inefficient. The present inductively defined data types do not improve that, and it might be interesting to study how that could be improved. Finally, the fact that the induction principle is not derivable has been repaired in a different way in the Cedille system [12], where also an analysis of Mendler style impredicative coinductive types has been made [19]. A detailed comparison of our approach with Cedille would be interesting.

References

1 Michael Gordon Abbott, Thorsten Altenkirch, and Neil Ghani. Containers: Constructing strictly positive types. *Theoretical Computer Science*, 342(1):3–27, 2005. `doi:10.1016/J.TCS.2005.06.002`.

2 Benedikt Ahrens, Paolo Capriotti, and Régis Spadotti. Non-wellfounded trees in homotopy type theory. In Thorsten Altenkirch, editor, *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland*, volume 38 of *LIPIcs*, pages 17–30. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. `doi:10.4230/LIPICS.TLCA.2015.17`.

3 Thorsten Altenkirch, Yorgo Chamoun, Ambrus Kaposi, and Michael Shulman. Internal parametricity, without an interval. *Proc. ACM Program. Lang.*, 8(POPL):2340–2369, 2024. `doi:10.1145/3632920`.

4 Steve Awodey, Jonas Frey, and Sam Speight. Impredicative encodings of (higher) inductive types. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 76–85. ACM, 2018. `doi:10.1145/3209108.3209130`.

5 Steven Awodey, Nicola Gambino, and Kristina Sojakova. Inductive types in homotopy type theory. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 95–104. IEEE Computer Society, 2012. `doi:10.1109/LICS.2012.21`.

6 Henk Barendregt. Introduction to generalized type systems. *Journal of Functional Programming*, 1(2):125–154, 1991. `doi:10.1017/S0956796800020025`.

7 Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of agda - A functional language with dependent types. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, volume 5674 of *Lecture Notes in Computer Science*, pages 73–78. Springer, 2009. `doi:10.1007/978-3-642-03359-9_6`.

8 Alonzo Church. A formulation of the simple theory of types. *J. Symb. Log.*, 5(2):56–68, 1940. doi:10.2307/2266170.

9 Thierry Coquand and Gérard P. Huet. The Calculus of Constructions. *Inf. Comput.*, 76(2/3):95–120, 1988. doi:10.1016/0890-5401(88)90005-3.

10 Thierry Coquand and Christine Paulin. Inductively defined types. In Per Martin-Löf and Grigori Mints, editors, *COLOG-88, International Conference on Computer Logic, Tallinn, USSR, December 1988, Proceedings*, volume 417 of *Lecture Notes in Computer Science*, pages 50–66. Springer, 1988. doi:10.1007/3-540-52335-9_47.

11 Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer, 2021. doi:10.1007/978-3-030-79876-5_37.

12 Denis Firsov and Aaron Stump. Generic derivation of induction for impredicative encodings in Cedille. In June Andronick and Amy P. Felty, editors, *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, pages 215–227. ACM, 2018. doi:10.1145/3167087.

13 Herman Geuvers. Induction is not derivable in second order dependent type theory. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications, 5th International Conference, TLCA 2001, Krakow, Poland*, volume 2044 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2001. doi:10.1007/3-540-45413-6_16.

14 Herman Geuvers. The Church-Scott representation of inductive and coinductive data, 2014. URL: https://api.semanticscholar.org/CorpusID:13897083.

15 Jean-Yves Girard. Une extension de l'interpretation de Gödel a l'analyse, et son application a l'elimination des coupures dans l'analyse et la theorie des types. In *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92. Elsevier, 1971. doi:10.1016/S0049-237X(08)70843-7.

16 Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*. Cambridge University Press, 1993.

17 Claudio Hermida and Bart Jacobs. Structural induction and coinduction in a fibrational setting. *Information and Computation*, 145(2):107–152, 1998. doi:10.1006/inco.1998.2725.

18 Bart Jacobs and Jan Rutten. An introduction to (co)algebra and (co)induction. In *Advanced Topics in Bisimulation and Coinduction*, volume 52 of *Cambridge tracts in theoretical computer science*, pages 38–99. Cambridge University Press, 2012.

19 Christopher Jenkins, Aaron Stump, and Larry Diehl. Efficient lambda encodings for Mendler-style coinductive types in Cedille. In Max S. New and Sam Lindley, editors, *Proceedings Eighth Workshop on Mathematically Structured Functional Programming, MSFP@ETAPS 2020, Dublin, Ireland, 25th April 2020*, volume 317 of *EPTCS*, pages 72–97, 2020. doi:10.4204/EPTCS.317.5.

20 Neelakantan R. Krishnaswami and Derek Dreyer. Internalizing relational parametricity in the extensional calculus of constructions. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 432–451. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPICS.CSL.2013.432.

21 Nuo Li. *Quotient Types in Type Theory*. PhD thesis, University of Nottingham, 2014.

22 Per Martin-Löf, 1984. Notes by Giovanni Sambin of a series of lectures in Padua, 1980.

23 Rob Nederpelt and Herman Geuvers. *Type Theory and Formal Proof: An Introduction*. Cambridge University Press, 2014.

24 nLab authors. hom-functor preserves limits. https://ncatlab.org/nlab/show/hom-functor+preserves+limits, June 2024. Revision 11.

25 Daniël Otten. M-types and bisimulation. Bachelor's thesis, Leiden University, 2020. URL: https://theses.liacs.nl/pdf/2019-2020-OttenDD.pdf.

26 Christine Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 1993. doi:10.1007/BFB0037116.

27 Frank Pfenning and Christine Paulin-Mohring. Inductively defined types in the Calculus of Constructions. In *Mathematical Foundations of Programming Semantics, 5th International Conference, Tulane University, New Orleans, Louisiana, USA, March 29 - April 1, 1989, Proceedings*, volume 442 of *Lecture Notes in Computer Science*, pages 209–228. Springer, 1989. doi:10.1007/BFB0040259.

28 Felix Rech. *Strictly Positive Types in Homotopy Type Theory*. Bachelor Thesis, Saarland University, 2017.

29 John C. Reynolds. Towards a theory of type structure. In *Programming Symposium, Proceedings Colloque sur la Programmation, Paris, France, April 9-11, 1974*, volume 19 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1974. doi:10.1007/3-540-06859-7_148.

30 Egbert Rijke. Introduction to Homotopy Type Theory, 2022. arXiv:2212.11082.

31 Xavier Ripoll Echeveste. Alternative impredicative encodings of inductive types. Master's thesis, Universiteit van Amsterdam, 2023.

32 Ivar Rummelhoff. Polynat in PER models. *Theor. Comput. Sci.*, 316(1):215–224, 2004. doi:10.1016/J.TCS.2004.01.031.

33 Sam Speight. Impredicative encodings of inductive types in Homotopy Type Theory. Master's thesis, Carnegie Mellon University, Pittsburgh, USA, 2017. URL: https://www.cs.ox.ac.uk/people/sam.speight/publications/sams-hott-thesis.pdf0.

34 Rocq Development Team. The Rocq proof assistant, June 2024. doi:10.5281/zenodo.11551307.

35 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

36 Benno van den Berg and Federico De Marchi. Non-well-founded trees in categories. *Ann. Pure Appl. Logic*, 146(1):40–59, 2007. doi:10.1016/J.APAL.2006.12.001.

37 Philip Wadler. The Girard-Reynolds isomorphism. *Inf. Comput.*, 186(2):260–284, 2003. doi:10.1016/S0890-5401(03)00141-X.

38 Philip Wadler. The Girard-Reynolds isomorphism (second edition). *Theoretical Computer Science*, 375(1-3):201–226, 2007. doi:10.1016/J.TCS.2006.12.042.

A Inference rules of the system

These rules are partly taken from Appendix A of the master's thesis of Sam Speight [33] and similar rules appear in HoTT [35]. It can also be seen as the Calculus of Constructions, possible extended with an infinite hierarchy of universes $\mathcal{U}_i$ ($i \in \mathbb{N}$), where we write $\mathcal{U}$ for $\mathcal{U}_0$.

$$\frac{\Gamma \vdash A : \mathcal{U}_i \quad \Gamma, a : A \vdash B : \mathcal{U}}{\Gamma \vdash \Pi(a : A).B : \mathcal{U}}\ \Pi\text{-form}_1 \qquad \frac{\Gamma \vdash A : \mathcal{U}_i \quad \Gamma, a : A \vdash B : \mathcal{U}_j}{\Gamma \vdash \Pi(a : A).B : \mathcal{U}_{\max(i,j)}}\ \Pi\text{-form}_2$$

$$\frac{\Gamma, a : A \vdash b : B}{\Gamma \vdash \lambda(a : A).b : \Pi(a : A).B}\ \Pi\text{-intro} \qquad \frac{\Gamma \vdash f : \Pi(x : A).B \quad \Gamma \vdash a : A}{\Gamma \vdash f\ a : B[x := a]}\ \Pi\text{-elim}$$

$$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash a : A}{\Gamma \vdash (\lambda x.b)\ a \overset{\beta}{=} b[x := a] : B[x := a]}\ \Pi\text{-}\beta \qquad \frac{\Gamma \vdash f : \Pi(x : A).B}{\Gamma \vdash f \overset{\eta}{=} (\lambda x.f\ x) : \Pi(x : A).B}\ \Pi\text{-}\eta$$

$$\frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma, a : A \vdash B : \mathcal{U}}{\Gamma \vdash \Sigma(a : A).B : \mathcal{U}}\ \Sigma\text{-form}_1 \qquad \frac{\Gamma \vdash A : \mathcal{U}_i \quad \Gamma, a : A \vdash B : \mathcal{U}_j}{\Gamma \vdash \Sigma(a : A).B : \mathcal{U}_{\max(i,j)}}\ \Sigma\text{-form}_1$$

$$\frac{\Gamma, x : A \vdash B : \mathcal{U}_i \quad \Gamma \vdash a : A \quad \Gamma \vdash b : B[x := a]}{\Gamma \vdash \langle a, b\rangle : \Sigma(x : A).B}\ \Sigma\text{-intro}$$

$$\frac{\Gamma \vdash p : \Sigma(x : A).B}{\Gamma \vdash \mathtt{pr1}\ p : A}\ \Sigma\text{-elim} \qquad \frac{\Gamma \vdash p : \Sigma(x : A).B}{\Gamma \vdash \mathtt{pr2}\ p : B[x := \mathtt{pr1}\ p]}\ \Sigma\text{-elim}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma, a : A \vdash b : B}{\Gamma \vdash \mathtt{pr1}\ \langle a, b\rangle \overset{\beta}{=} a : A}\ \Sigma\text{-}\beta \qquad \frac{\Gamma \vdash a : A \quad \Gamma, x : A \vdash b : B}{\Gamma \vdash \mathtt{pr2}\ \langle a, b\rangle \overset{\beta}{=} b : B[x := a]}\ \Sigma\text{-}\beta$$

$$\frac{\Gamma \vdash p : \Sigma(x : A).B}{\Gamma \vdash p \overset{\eta}{=} \langle \mathtt{pr1}\ p,\ \mathtt{pr2}\ p\rangle : \Sigma(x : A).B}\ \Sigma\text{-}\eta \qquad \frac{\Gamma \vdash A : \mathcal{U}_i \quad \Gamma \vdash a : A}{\Gamma \vdash \mathtt{refl} : a =_A a}\ =\text{-intro}$$

$$\frac{\Gamma \vdash A : \mathcal{U} \quad \Gamma \vdash a : A \quad \Gamma \vdash b : A}{\Gamma \vdash (a =_A b) : \mathcal{U}}\ =\text{-form}_1 \qquad \frac{\Gamma \vdash A : \mathcal{U}_i \quad \Gamma \vdash a : A \quad \Gamma \vdash b : A}{\Gamma \vdash (a =_A b) : \mathcal{U}_i}\ =\text{-form}_1$$

$$\frac{\begin{array}{c}\Gamma \vdash a : A \qquad \Gamma \vdash b : B \qquad \Gamma \vdash p : a =_A b \\ \Gamma, x : A, y : A, p : a =_A b \vdash C : \mathcal{U}_i \qquad \Gamma, z : A \vdash c : C[z, z, x := \mathtt{refl}_z, y, p]\end{array}}{\Gamma \vdash \mathtt{ind}_=(c, a, b, q) : C[a, b, x := q, y, p]}\ =\text{-elim}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma, x : A, y : A, p : a =_A b \vdash C : \mathcal{U}_i \qquad \Gamma, z : A \vdash c : C[z, z, x := \mathtt{refl}_z, y, p]}{\Gamma \vdash \mathtt{ind}_=(c, a, a, \mathtt{refl}_a) \overset{\beta}{=} c[z := a] : C[a, a, x := \mathtt{refl}_a, y, p]}\ =\text{-}\beta$$

$$\begin{aligned}\mathtt{FunExt} &: \Pi(f, g : \Pi(x : X), Y\ x).(\Pi(x : X).f\ x = g\ x) \implies f = g \\ \mathtt{UIP} &: \Pi(X : \mathcal{U}).\Pi(x, y : X).\Pi(p, q : x = y).p = q\end{aligned}$$

B Assorted proofs

▶ **Lemma B.1** (Lemma 4.13). *The stream recursor, together with the head function* $\mathtt{hd}$ *and the tail function* $\mathtt{tl}$ *form the identity.*

$$\mathit{corec}_s\ \mathit{Stream}\ \mathtt{hd}\ \mathtt{tl} = \mathit{id}_{\mathit{Stream}}$$

Proof. By Lemma 3.12, we know that $\overline{\mathtt{cls}} = \mathtt{id}_{\mathtt{Stream}}$, where $\overline{\mathtt{cls}} := \mathtt{rec_q}\ \mathtt{Stream}\ \mathtt{cls}\ \mathtt{EqCls2}$. If we then apply the η-rule of the quotient type (Theorem 3.13), we see that, in order to show that $\mathtt{corec_s}\ \mathtt{Stream}\ \mathtt{hd}\ \mathtt{tl} = \mathtt{id}_{\mathtt{Stream}} = \mathtt{rec_q}\ \mathtt{Stream}\ \mathtt{cls}\ \mathtt{EqCls2}$, it is enough to show the following equation:

$$(\mathtt{corec_s}\ \mathtt{Stream}\ \mathtt{hd}\ \mathtt{tl}) \circ \mathtt{cls} = \mathtt{cls} \tag{7}$$

We reduce Equation (7) using function extensionality, the definition of $\mathtt{corec_s}$.

$$\begin{aligned} & \mathtt{corec_s}\ \mathtt{Stream}\ \mathtt{hd}\ \mathtt{tl} \circ \mathtt{cls} = \mathtt{cls} && \overset{\text{FunExt}}{\Longleftrightarrow} \\ \forall (s : \mathtt{Stream}^*) \quad & \mathtt{corec_s}\ \mathtt{Stream}\ \mathtt{hd}\ \mathtt{tl}\ (\mathtt{cls}\ s) = \mathtt{cls}\ s && \Longleftrightarrow \\ \forall (s : \mathtt{Stream}^*) \quad & \mathtt{cls}\ (\mathtt{corec_s^*}\ \mathtt{Stream}\ \mathtt{hd}\ \mathtt{tl}\ (\mathtt{cls}\ s)) = \mathtt{cls}\ s && \end{aligned} \tag{8}$$

We shall prove Equation (8). Assume a stream $(s : \mathtt{Stream}^*)$. We use the existential recursor $\mathtt{rec}_\exists$ and Lemma 4.3 to destruct s. We thus destruct $(s = \mathtt{corec_s^*}\ X\ h\ t\ x)$ for some $(X : \mathcal{U})$, $(h : X \to E)$, $(t : X \to X)$ and $(x : X)$. Now it follows from the following equalities.

$$\begin{aligned} \mathtt{cls}\ (\mathtt{corec_s^*}\ \mathtt{Stream}\ \mathtt{hd}\ \mathtt{tl}\ (\mathtt{cls}\ s)) &\overset{4.11}{=} \mathtt{cls}\ (\mathtt{corec_s^*}\ \mathtt{Stream}^*\ \mathtt{hd}^*\ \mathtt{tl}^*\ s) \\ &\overset{4.12}{=} \mathtt{cls}\ s \end{aligned}$$

◀

▶ **Theorem 2** (Theorem 4.16). *We have the coinduction proof principle, which means that for all* $\sigma, \tau : \mathit{Stream}$*, we have* $\sigma \sim \tau$ *if and only if* $\sigma = \tau$*.*

Proof. ($\Longleftarrow$) follows because $=$ is a bisimulation.

($\Longrightarrow$) Let $(\sigma, \tau : \mathtt{Stream})$. Assume we have $\sigma \sim \tau$. Consider the following quotient $\mathtt{Stream}/{\sim} := \mathtt{quot}\ \mathtt{Stream}\ \mathtt{BiSim}$. From $\sigma \sim \tau$ we know that there exists a bisimulation $(R : \mathtt{Stream} \to \mathtt{Stream} \to \mathcal{U})$ with $(R\ \sigma\ \tau)$ and $(\mathtt{IsBiSim}\ R)$ so we have $\mathtt{hd}\ \sigma = \mathtt{hd}\ \tau$ and $R\ (\mathtt{tl}\ \sigma)\ (\mathtt{tl}\ \tau)$. This implies that $\langle \overline{\mathtt{hd}}, \overline{\mathtt{cls}_\sim \circ \mathtt{tl}} \rangle : \mathtt{Stream}/{\sim} \to E \times \mathtt{Stream}/{\sim}$ forms an $\mathcal{S}$-coalgebra and we have $\mathtt{corec_s}\ (\mathtt{Stream}/{\sim})\ \overline{\mathtt{hd}}\ \overline{\mathtt{cls}_\sim \circ \mathtt{tl}} : \mathtt{Stream}/{\sim} \to \mathtt{Stream}$. We define the function $f : \mathtt{Stream}/{\sim} \to \mathtt{Stream}$ as follows.

$$f := \mathtt{corec_s}\ (\mathtt{Stream}/{\sim})\ \overline{\mathtt{hd}}\ \overline{\mathtt{cls}_\sim \circ \mathtt{tl}}.$$

Now, $f \circ \mathtt{cls}_\sim : \mathtt{Stream} \to \mathtt{Stream}$ is a $\mathcal{S}$-coalgebra morphism and by the η-rule for streams (Theorem 4.14), we find that $f = \mathtt{id}_{\mathtt{Stream}}$. Therefore we have

$$\sigma = f(\mathtt{cls}_\sim\ \sigma) = f(\mathtt{cls}_\sim\ \tau) = \tau.$$

◀

▶ **Lemma B.3** (Lemma 7.4). *The right projection of the M-eliminator* elimM^* *is constant on all equivalence classes of* CoLimM*.*

$$\Pi(m, n : M_{AB}).m \equiv n \implies (\mathit{cls}\ (\mathit{pr2}\ (\mathit{elimM}^*\ m))) = (\mathit{cls}\ (\mathit{pr2}\ (\mathit{elimM}^*\ n)))$$

Proof. Let $(m, n : M_{AB})$ be M-types. Suppose that $(\mathtt{CoLimM}\ m\ n)$ holds. We thus have some $(X, Y : \mathcal{U})$, $(g : X \to \Sigma(a : A).B(a) \to X)$, $(g' : Y \to \Sigma(a : A).B(a) \to Y)$, $(x : X)$, and $(f : X \to Y)$ such that the following hold:

$$\mathtt{pr1}\ (g'\ (f\ x)) = \mathtt{pr1}\ (g\ x)$$
$$\mathtt{pr2}\ (g'\ (f\ x)) = f \circ (\mathtt{pr2}\ (g\ x)) \tag{9}$$
$$m = \mathtt{corec}^*_{\mathtt{M}}\ X\ g\ x \tag{10}$$
$$n = \mathtt{corec}^*_{\mathtt{M}}\ Y\ g'\ (f\ x) \tag{11}$$

We derive the following. Here we use an equality, indicated by $(*)$, that we prove next.

$$\begin{aligned}
\mathtt{cls}\ (\mathtt{pr2}\ (\mathtt{elimM}^*\ m)) &\overset{9}{=} \mathtt{cls}\ (\mathtt{pr2}\ (\mathtt{elimM}^*\ (\mathtt{corec}^*_{\mathtt{M}}\ X\ g\ x))) \\
&= \mathtt{cls}\ (\mathtt{pr2}\ \langle \mathtt{pr1}\ (g\ x),\ \lambda b.\mathtt{corec}^*_{\mathtt{M}}\ X\ g\ (\mathtt{pr2}\ (g\ x)\ b)\rangle) \\
&\overset{\beta}{=} \mathtt{cls}\ (\lambda b.\mathtt{corec}^*_{\mathtt{M}}\ X\ g\ (\mathtt{pr2}\ (g\ x)\ b)) \\
&\overset{(*)}{=} \mathtt{cls}\ (\lambda b.\mathtt{corec}^*_{\mathtt{M}}\ Y\ g'\ (f\ (\mathtt{pr2}\ (g\ x)\ b))) \\
&\overset{9}{=} \mathtt{cls}\ (\lambda b.\mathtt{corec}^*_{\mathtt{M}}\ Y\ g'\ (\mathtt{pr2}\ (g'\ (f\ x)))) \\
&\overset{\beta}{=} \mathtt{cls}\ (\mathtt{pr2}\ \langle \mathtt{pr1}\ (g'\ (f\ x)),\ \lambda b.\mathtt{corec}^*_{\mathtt{M}}\ Y\ g'\ (\mathtt{pr2}\ (g'\ (f\ x))\ b)\rangle) \\
&= \mathtt{cls}\ (\mathtt{pr2}\ (\mathtt{elimM}^*\ (\mathtt{corec}^*_{\mathtt{M}}\ Y\ g'\ (f\ x)))) \\
&\overset{11}{=} \mathtt{cls}\ (\mathtt{pr2}\ (\mathtt{elimM}^*\ n))
\end{aligned}$$

We shall now fill in the step $(*)$. Let $(b : B(a))$. It suffices to show the following:

$$\mathtt{corec}^*_{\mathtt{M}}\ X\ g\ (\mathtt{pr2}\ (g\ x)\ b) \equiv \mathtt{corec}^*_{\mathtt{M}}\ Y\ g'\ (f\ (\mathtt{pr2}\ (g\ x)\ b)) \tag{12}$$

This clearly holds since f is indeed a morphism by assumption. ◀

▶ **Lemma B.4** (Lemma 7.6). *We have $m \equiv (\mathit{corec}^*_M\ M^*_{AB}\ \mathit{elimM}^*\ m)$ for each $(m : M^*_{AB})$.*

Proof. We destruct m to obtain $(X : \mathcal{U})$, $(x : X)$ and $(g : X \to \Sigma(a : A).B(a) \to X)$ such that $m = \mathtt{pack}\ X\ \langle x, g\rangle =: \mathtt{corec}^*_{\mathtt{M}}\ X\ g\ x$ by using Lemma 4.3. In addition, we have $m = \mathtt{corec}^*_{\mathtt{M}}\ X\ g\ x$ and $\mathtt{corec}^*_{\mathtt{M}}\ M^*_{AB}\ \mathtt{elimM}^*\ m = \mathtt{corec}^*_{\mathtt{M}}\ M^*_{AB}\ \mathtt{elimM}^*\ ((\mathtt{corec}^*_{\mathtt{M}}\ X\ g)\ x)$ since $(\mathtt{corec}^*_{\mathtt{M}}\ X\ g)$ is an $\mathcal{M}$-morphism, Hence, we have that $m \equiv (\mathtt{corec}^*_{\mathtt{M}}\ M^*_{AB}\ \mathtt{elimM}^*\ m)$. ◀

▶ **Lemma B.5** (Lemma 7.7). *We have $\mathit{corec}_M\ M_{AB}\ \mathit{elimM} = \mathit{id}_{M_{AB}}$.*

Proof. By functional extensionality, it suffices to prove that

$$\mathtt{cls}\ (\mathtt{corec}^*_{\mathtt{M}}\ M_{AB}\ \mathtt{elimM}\ (\mathtt{cls}\ m)) = \mathtt{cls}\ m \text{ for each } m : M^*_{AB}.$$

We destruct $m = \mathtt{corec}^*_{\mathtt{M}}\ X\ g\ x$ for some $(X : \mathcal{U})$, $(g : X \to \Sigma(a : A).B(a) \to X)$ and $(x : X)$. We conclude the following.

$$\mathtt{cls}\ (\mathtt{corec}^*_{\mathtt{M}}\ M_{AB}\ \mathtt{elimM}\ (\mathtt{cls}\ m)) = \mathtt{cls}\ (\mathtt{corec}^*_{\mathtt{M}}\ M^*_{AB}\ \mathtt{elimM}^*\ m) = \mathtt{cls}\ m$$ ◀