Theoretical Informatics and Applications Will be set by the publisher
Informatique Théorique et Applications

TWO EXTENSIONS OF SYSTEM F WITH (CO)ITERATION AND PRIMITIVE (CO)RECURSION PRINCIPLES *

Favio Ezequiel Miranda-Perea[1]

Abstract. This paper presents two extensions of the second order polymorphic lambda calculus F with inductive and coinductive types including not only (co)iteration but primitive (co)recursion and inversion principles. The systems are proven to be safe, can be seen as extensions of Hagino's categorical lambda calculus and are also related with the systems of higher-order iterators of [1].

1991 Mathematics Subject Classification. 03B40, 68N18, 68Q42, 68Q65.

1. Introduction

The main goal of this paper is to fill a gap between two kinds of type systems involving inductive and coinductive types, namely the lambda calculus with categorical type constructors of [7] and the higher order systems for (co)iteration developed in [1]. In [7] Hagino develops an strongly normalising extension of simply typed lambda calculus with clausular[1] and positive inductive and coinductive types and iteration principles called there categorical type constructors, modelling the concept of (weak) initial and final dialgebras of functors, concept which is also introduced there. Later, in [25], this system called there $C\lambda$, is encoded into system F, apparently defining for the first time the now usual code for coinductive types. On the other hand [1] presents several systems of (co)inductive constructors of higher kinds (higher-order nested datatypes) which all happen to be definable within the system F^ω of higher-order parametric polymorphism. So we have an extension of simply typed lambda calculus with (co)inductive types definable in system F and several extensions of system F^ω with (co)inductive constructors, which are definable in F^ω. To our knowledge, the taxonomy of type systems lacks of a system combining (co)inductive types, polymorphism and primitive (co)recursion. Though there exist extensions of system F, in Church style, with both inductive and coinductive types including primitive (co)recursion principles (see [11, 12]), these systems handle either inductive or coinductive types but not both at the same time. In this paper we present two type systems handling both inductive and coinductive types as well as polymorphism and primitive (co)recursion. One system handles conventional (co)inductive types modellin (co)algebras whereas the other models dialgebras, as in [7], using clausular types. Both systems are safe, for their operational semantics terminates and preserves types. Moreover, the systems are full monotone, that is, there is no positivity restriction in constructing a (co)inductive type.

1.1. Overview of the paper

After mentioning some preliminaries on categories and polymorphic lambda calculus, we develop our first system called MICT, develop some examples of programming and directly prove the termination (strong normalization) of the operational semantics. In section 4 we present the second system which

Keywords and phrases: Coiteration, corecursion, iteration, primitive recursion, System F, monotone inductive type, monotone coinductive type, monotonicity witness, algebras, coalgebras, dialgebras

* *This research has been supported by Conacyt-UNAM Mexico postdoctoral grant number 50289*

[1] Departamento de Matematicas, Facultad de Ciencias UNAM Circuito Exterior S/N. Ciudad Universitaria 04510 Mexico D.F. Mexico. E-mail: favio@ciencias.unam.mx

[1] The name is mine

enhaces the former by allowing definitions with several constructors/destructors, feature which is illustrated in several examples. Safety for this system is proven by embedding it into the previous system to ensure termination and by proving directly its type-preservation, a non-trivial property due to the use of the Curry-style formalism. Finally we point to some future and related work.

2. Preliminaries

In this section we recall some categorical concepts as well as our base type system, the second order polymorphic lambda calculus F.

2.1. System F

Our basic framework is the well-known system F of Girard and Reynolds in Curry-style presentation. For ease of presentation we include sum and product types as primitive constructors.

- Types built from an infinite set of type variables denoted by X.

$$A, B, C, F, G ::= X \mid A \to B \mid \forall X A \mid A + B \mid A \times B$$

- Terms built from an infinite set of term variables denoted by x.

$$t, r, s \quad ::= \quad x \mid \lambda x r \mid rs \mid \mathsf{inl}\, r \mid \mathsf{inr}\, s \mid \mathsf{case}(r, x.s, y.t) \mid \langle r, s\rangle \mid \mathsf{fst}\, r \mid \mathsf{snd}\, r$$

- Contexts are sets of the form $\Gamma = \{x_1 : A_1, \ldots, x_n : A_n\}$. The expression $\Gamma, x : A$ denotes the context $\Gamma \cup \{x : A\}$ always assuming that x was not previously declared in Γ.
- Typing rules of the form $\Gamma \vdash t : A$ denoting that t is a wellformed term of type A in context Γ.

$$\Gamma, x : A \vdash x : A \;\; (Var)$$

$$\frac{\Gamma, x : A \vdash r : B}{\Gamma \vdash \lambda x r : A \to B}\,(\to I) \quad \frac{\Gamma \vdash r : A \to B \quad \Gamma \vdash s : A}{\Gamma \vdash rs : B}\,(\to E)$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X A}\,(\forall I) \quad \frac{\Gamma \vdash t : \forall X A}{\Gamma \vdash t : A[X := F]}\,(\forall E)$$

$$\frac{\Gamma \vdash r : A}{\Gamma \vdash \mathsf{inl}\, r : A + B}\;(+I_L) \qquad \frac{\Gamma \vdash r : B}{\Gamma \vdash \mathsf{inr}\, r : A + B}\;(+I_R)$$

$$\frac{\Gamma \vdash r : A + B \quad \Gamma, x : A \vdash s : C \quad \Gamma, y : B \vdash t : C}{\Gamma \vdash \mathsf{case}(r, x.s, y.t) : C}\;(+E)$$

$$\frac{\Gamma \vdash r : A \quad \Gamma \vdash s : B}{\Gamma \vdash \langle r, s\rangle : A \times B}\;(\times I) \qquad \frac{\Gamma \vdash s : A \times B}{\Gamma \vdash \mathsf{fst}\, s : A}\;(\times E_L) \qquad \frac{\Gamma \vdash s : A \times B}{\Gamma \vdash \mathsf{snd}\, s : B}\;(\times E_R)$$

- Reduction. The operational semantics is given by the one-step β-reduction relation $t \to t'$ defined as the closure of the following axioms under all term formers.

$$\begin{array}{rcl}
(\lambda x r)s & \mapsto_\beta & r[x := s] \\
\mathsf{case}(\mathsf{inl}\, r, x.s, y.t) & \mapsto_\beta & s[x := r] \\
\mathsf{case}(\mathsf{inr}\, r, x.s, y.t) & \mapsto_\beta & t[y := r] \\
\mathsf{fst}\langle r, s\rangle & \mapsto_\beta & r \\
\mathsf{snd}\langle r, s\rangle & \mapsto_\beta & s
\end{array}$$

2.2. Algebras and Coalgebras

We assume some knowledge of category theory, here we only state the basic concepts needed later, for full details on category theory see for example [10].
We will use the categorical approach to (co)induction (see [9]) to formulate our systems of (co)inductive types, this can be briefly stated as follows:

- Induction is the use of initiality for algebras
- Coinduction is the use of finality for coalgebras

Fix a category $\mathcal{C}$, with products and coproducts for our purposes.

Definition 1. *Let $T : \mathcal{C} \to \mathcal{C}$ be a functor. A T-algebra is a pair $\langle A, f\rangle$ such that $f : TA \to A$. Analogously a T-coalgebra is a pair $\langle B, g\rangle$ with $g : B \to TB$.*

Definition 2. *Given two T-algebras $\langle A, f\rangle, \langle B, g\rangle$ a morphism from $\langle A, f\rangle$ to $\langle B, g\rangle$ is a $\mathcal{C}$-morphism $h : A \to B$ such that the following diagram commutes:*

$$\begin{array}{ccc} TA & \xrightarrow{f} & A \\ {\scriptstyle Th}\downarrow & & \downarrow{\scriptstyle h} \\ TB & \xrightarrow{g} & B \end{array}$$

We say that the algebra $\langle A, f\rangle$ is initial if it is the initial object of the category of T-algebras, i.e., if for every given algebra $\langle B, g\rangle$ there is a unique h such that the above diagram commutes, in this case the h is denoted It_g and called the iteratively defined morphism with step function g.
If exists, the initial T-algebra is unique and is denoted as $\langle \mu T, \mathsf{in}_T\rangle$, so that $\mathsf{It}_g : \mu T \to B$ and

$$\mathsf{It}_g \circ \mathsf{in}_T = g \circ T(\mathsf{It}_g) \tag{1}$$

this equation is called principle of iteration.
Dually a morphism of coalgebras from $\langle B, g\rangle$ to $\langle A, f\rangle$ is a $\mathcal{C}$-morphism $h : B \to A$ such that the following diagram commutes:

$$\begin{array}{ccc} A & \xrightarrow{f} & TA \\ {\scriptstyle h}\uparrow & & \uparrow{\scriptstyle Th} \\ B & \xrightarrow{g} & TB \end{array}$$

We say that the coalgebra $\langle A, f\rangle$ is final if it is the final object of the category of T-algebras, i.e., if for every given coalgebra $\langle B, g\rangle$ there is a unique h such that the above diagram commutes., in this case we denote such h with Colt_g and call it the coiteratively defined morphism with step function g.
If exists, the final T-coalgebra is unique and denoted with $\langle \nu T, \mathsf{out}_T\rangle$, so that $\mathsf{Colt}_g : B \to \nu T$ and

$$\mathsf{out}_T \circ \mathsf{Colt}_g = F(\mathsf{Colt}_g) \circ g \tag{2}$$

this equation is called principle of coiteration.

Proposition 1. *$\mathsf{in}_T, \mathsf{out}_T$ are isomorphisms, therefore there exist inverse morphisms ${\mathsf{in}_T}^{-1}, {\mathsf{out}_T}^{-1}$ such that ${\mathsf{in}_T}^{-1} \circ \mathsf{in}_T = \mathsf{Id}_{T\mu T}$ and $\mathsf{out}_T \circ {\mathsf{out}_T}^{-1} = \mathsf{Id}_{\nu T}$. These equations are called the principle of inductive and coinductive inversion respectively.*

Proof. Straightforward. □

The extended (co)induction principles will be justified by means of (co)recursive algebras:

Definition 3. *Define $\Pi_D : \mathcal{C} \to \mathcal{C}$ as $\Pi_D C := C \times D$. We say that the T-algebra $\langle A, f\rangle$ is recursive if for every $T\Pi_A$-algebra $\langle B, g\rangle$ there exists a morphism $h : A \to B$ such that:*

$$\begin{array}{ccc} TA & \xrightarrow{f} & A \\ {\scriptstyle T\langle \mathsf{Id}, h\rangle}\downarrow & & \downarrow{\scriptstyle h} \\ T(A \times B) & \xrightarrow{g} & B \end{array} \tag{3}$$

Set $\Sigma_D : \mathcal{C} \to \mathcal{C}$ with $\Sigma_D C := C + D$. We say that the T-coalgebra $\langle A, f\rangle$ is corecursive if for every $T\Sigma_A$-coalgebra $\langle B, g\rangle$ there exists a morphism $h : B \to A$ such that:

$$\begin{array}{ccc} A & \xrightarrow{f} & TA \\ {\scriptstyle h}\uparrow & & \uparrow{\scriptstyle T[\mathsf{Id}, h]} \\ B & \xrightarrow{g} & T(A + B) \end{array} \tag{4}$$

Proposition 2. *$\langle \mu T, \mathsf{in}_T\rangle$ is recursive and $\langle \nu T, \mathsf{out}_T\rangle$ is corecursive.*

Proof. Straightforward. □

For the cases of the initial algebra and the final coalgebra, the h that makes diagrams (3), (4) commute is denoted $\mathsf{Rec}_g, \mathsf{CoRec}_g$ respectively and we refer to them as the (co)recursively defined morphism with step function g, so that we have $\mathsf{Rec}_g : \mu T \to B$, $\mathsf{CoRec}_g : B \to \nu T$ such that the following principles hold:

- Principle of Primitive Recursion

$$\mathsf{Rec}_g \circ \mathsf{in}_T = g \circ T(\langle \mathsf{Id}, \mathsf{Rec}_g\rangle) \tag{5}$$

- Principle of Primitive Corecursion

$$\mathsf{out}_T \circ \mathsf{CoRec}_g = T([\mathsf{Id}, \mathsf{CoRec}_g]) \circ g \tag{6}$$

2.3. Dialgebras

The concept of dialgebra, introduced in [8], is a straightforward generalization of (co)algebras with stronger expressive power (see [20]). With dialgebras we can represent products, coproducts and even exponential objects (see [4]). We will serve later from this concept to justify the clausular feature of one type system.

Definition 4. *Let $F, G : \mathcal{C} \to \mathcal{D}$ covariant functors between two categories $\mathcal{C}, \mathcal{D}$. A F, G-dialgebra is a pair $\langle A, f\rangle$ where A is a $\mathcal{C}$-object and $f : FA \to GA$ is a D-morphism.*

Definition 5. *A morphism between two F, G-dialgebras $\langle A, f\rangle, \langle B, g\rangle$ is a $\mathcal{C}$-morphism $h : A \to B$ such that:*

$$\begin{array}{ccc} FA & \xrightarrow{f} & GA \\ {\scriptstyle Fh}\downarrow & & \downarrow{\scriptstyle Gh} \\ FB & \xrightarrow{g} & GB \end{array}$$

Observe that if I is the identity functor then a T, I-dialgebra $\langle A, f\rangle$ is a T-algebra and an I, T-dialgebra is a T-coalgebra.

We are specially interested in dialgebras where the functors $F, G : \mathcal{C} \to \mathcal{C}^n$ are of the form

$$F \equiv \langle F_1, \ldots, F_n \rangle \qquad G \equiv \langle I, \ldots, I \rangle$$

with $F_i : \mathcal{C} \to \mathcal{C}$.

The final G, F-dialgebra, if exists, will be denoted with $\langle \nu(F_1, \ldots, F_n), \mathsf{out}_n \rangle$

The finality of $\nu(F_1, \ldots, F_n)$ is given by the following diagram, where $V := \nu(F_1, \ldots, F_n)$

$$\begin{array}{ccc} \langle V, \ldots, V \rangle & \xrightarrow{\mathsf{out}_n} & \langle F_1 V, \ldots, F_n V \rangle \\ \uparrow{\scriptstyle \langle h, \ldots, h \rangle} & & \uparrow{\scriptstyle \langle F_1 h, \ldots, F_n h \rangle} \\ \langle B, \ldots, B \rangle & \xrightarrow{g} & \langle F_1 B, \ldots, F_n B \rangle \end{array}$$

where $h : B \to V$ is the unique function such that:

$$\mathsf{out}_n \circ \langle h, \ldots, h \rangle = \langle F_1 h, \ldots, F_n h \rangle \circ g$$

Observing that the morphisms out_n, g are neccesarily of the form

$$\mathsf{out}_n = \langle \mathsf{out}_{n,1}, \ldots, \mathsf{out}_{n,n} \rangle \qquad g = \langle g_1, \ldots, g_n \rangle.$$

The previous diagram can be splitted into the following n-diagrams, denoting the unique h above with Colt^n_g.

$$\begin{array}{ccc} \nu(F_1, \ldots, F_n) & \xrightarrow{\mathsf{out}_{n,i}} & F_i\big(\nu(F_1, \ldots, F_n)\big) \\ \uparrow{\scriptstyle \mathsf{Colt}^n_g} & & \uparrow{\scriptstyle F_i(\mathsf{Colt}^n_g)} \\ B & \xrightarrow{\qquad g_i \qquad} & F_i B \end{array}$$

$$\mathsf{out}_{n,i} \circ \mathsf{Colt}^n_g = F_i(\mathsf{Colt}^n_g) \circ g_i \tag{7}$$

These equations represent the coiteration principle on dialgebras

Analogously corecursion is introduced by the following n-diagrams :

$$\begin{array}{ccc} \nu(F_1, \ldots, F_n) & \xrightarrow{\mathsf{out}_{n,i}} & F_i\big(\nu(F_1, \ldots, F_n)\big) \\ \uparrow{\scriptstyle \mathsf{CoRec}^n_g} & & \uparrow{\scriptstyle F_i\big([\mathsf{Id}, \mathsf{CoRec}^n_g]\big)} \\ B & \xrightarrow{g_i} & F_i\big(\nu(F_1, \ldots, F_n) + B\big) \end{array}$$

$$\mathsf{out}_{n,i} \circ \mathsf{CoRec}^n_g = F_i\big([\mathsf{Id}, \mathsf{CoRec}^n_g]\big) \circ g_i \tag{8}$$

This equations represent the principle of primitive corecursion on dialgebras

Finally the coinductive inversion principle is given by this equations:

$$\mathsf{out}_k \circ \mathsf{out}_k^{-1} = \mathsf{Id}_{\langle F_1 V, \ldots, F_n V \rangle} \tag{9}$$

Similarly denoting with $\langle \mu(F_1,\ldots,F_n), \mathsf{in}_n\rangle$ the initial F,G-dialgebra we arrive to the following diagrams:

$$\begin{array}{ccc} F_i\big(\mu(F_1,\ldots,F_n)\big) & \xrightarrow{\mathsf{in}_{n,i}} & \mu(F_1,\ldots,F_n) \\ \Big\downarrow{\scriptstyle F_i(\mathsf{It}^n_g)} & & \Big\downarrow{\scriptstyle \mathsf{It}^n_g} \\ F_iB & \xrightarrow{\quad g_i \quad} & B \end{array}$$

representing the iteration principle:

$$\mathsf{It}^n_g \circ \mathsf{in}_{n,i} = g_i \circ F_i(\mathsf{It}^n_g) \tag{10}$$

$$\begin{array}{ccc} F_i\big(\mu(F_1,\ldots,F_n)\big) & \xrightarrow{\mathsf{in}_{n,i}} & \mu(F_1,\ldots,F_n) \\ \Big\downarrow{\scriptstyle F_i(\langle \mathsf{Id}, \mathsf{Rec}^n_g\rangle)} & & \Big\downarrow{\scriptstyle \mathsf{Rec}^n_g} \\ F_i\big(\mu(F_1,\ldots,F_n)\times B\big) & \xrightarrow{\quad g_i \quad} & B \end{array}$$

representing the recursion principle

$$\mathsf{Rec}^n_g \circ \mathsf{in}_{n,i} = g_i \circ F_i\big(\langle \mathsf{Id}, \mathsf{Rec}^n_g\rangle\big) \tag{11}$$

Finally the inductive inversion principle is given by:

$$\mathsf{in}_k^{-1} \circ \mathsf{in}_k = \mathsf{Id}_{\langle \mu(F_1,\ldots,F_n),\ldots,\mu(F_1,\ldots,F_n)\rangle} \tag{12}$$

2.4. From Categories to Types

Our goal is to define extensions of system F with type constructors for initial algebras and final coalgebras. To do this we need to see the type system as a category $\mathcal{C}$. Pragmatically, types will be objects, functions from one type to another will be morphisms, composition will be the usual function composition $g \circ f := \lambda z.g(fz)$, and the commutative diagrams presented above will generate the operational semantics which will allow us to program in the system. Such categories of types and its features are well-known, see for example [3], here we only assume its existence.

A functor $F : \mathcal{C} \to \mathcal{C}$ is then a transformation between types. We are specially interested in functors depending on a type variable X mapping a type B to a type $F[X := B]$, which will be denoted by λXF abstracting the type variable X. It is understood that $(\lambda XF)B$ means $F[X := B]$. Note that the systems developed in this paper are not higher-order and therefore abstractions like λXF will not be part of their official syntax. Our aim is to define type systems with (co)inductive types representing initial algebras and final coalgebras of functors λXF, denoted μXF, respectively νXF. Observe than an expression λXF is not immediately a functor because we only know its action on objects (types) but not on morphisms.

Type systems handling some kind of recursive or (co)inductive types usually require the syntactical condition of X ocurring only on positive positions in F to allow the construction of the types $\mu XF, \nu XF$ (See for example [5–7, 25]). Such condition guarantees the functoriality or monotonicity of F on function types. In our treatment we prefer to follow [11, 12] and use full monotonicity instead: the functoriality of λXF on morphisms is represented internally by means of a term $\mathsf{map} : F\,\mathsf{mon}\,X$ in a given context,

where its type, defined as $F\,\mathsf{mon}\,X := \forall X \forall Y.(X \to Y) \to F \to F[X ::= Y]$, expresses the fact that the functor λXF is monotone (covariant) with respect to X. Such terms are called *monotonicity witnesses*. Therefore a functor in our framework is a pair $\langle \lambda XF, \mathsf{map} \rangle$ where map is a term of type $F\,\mathsf{mon}\,X$ (in a given context). This way of defining functors is reminiscent of the way functors are defined in functional programming languages like Haskell, where this concept is captured by the following class definition:

```
class  Functor f  where
    fmap :: (a -> b) -> f a -> f b
```

Therefore a functor is not only a function `f` between categories but a pair composed of a function `f` and a mapping `fmap` who plays the role of the functor on morphisms.

3. A Type System for (Co)algebras

In this section we present the system MICT, which will model initial algebras and final coalgebras using inductive and coinductive types respectively.

3.1. Definition of the system

Extend system F as follows:

- Types:

$$A, B, C, F, G ::= \ldots \mid \mu XF \mid \nu XF$$

- Terms:

$$t, r, s, m \quad ::= \quad \ldots \mid \mathsf{It}(m, s, t) \mid \mathsf{Rec}(m, s, t) \mid \mathsf{in}\,t \mid \mathsf{CoIt}(m, s, t) \mid \mathsf{CoRec}(m, s, t) \mid \mathsf{out}\,t \mid \mathsf{out}^{-1}(m, t)$$

- Typing rules:

$$\frac{\Gamma \vdash t : F[X := \mu XF]}{\Gamma \vdash \mathsf{in}\,t : \mu XF} \ (\mu I)$$

$$\frac{\begin{array}{l}\Gamma \vdash t : \mu XF \\ \Gamma \vdash m : F\,\mathsf{mon}\,X \\ \Gamma \vdash s : F[X := B] \to B\end{array}}{\Gamma \vdash \mathsf{It}(m, s, t) : B} \ (\mu E)$$

$$\frac{\begin{array}{l}\Gamma \vdash t : \mu XF \\ \Gamma \vdash m : F\,\mathsf{mon}\,X \\ \Gamma \vdash s : F[X := \mu XF \times B] \to B\end{array}}{\Gamma \vdash \mathsf{Rec}(m, s, t) : B} \ (\mu E^{+})$$

$$\frac{\begin{array}{l}\Gamma \vdash s : B \to F[X := B] \\ \Gamma \vdash m : F\,\mathsf{mon}\,X \\ \Gamma \vdash t : B\end{array}}{\Gamma \vdash \mathsf{CoIt}(m, s, t) : \nu XF} \ (\nu I)$$

$$\frac{\begin{array}{l}\Gamma \vdash s : B \to F[X := \nu XF + B] \\ \Gamma \vdash m : F\,\mathsf{mon}\,X \\ \Gamma \vdash t : B\end{array}}{\Gamma \vdash \mathsf{CoRec}(m, s, t) : \nu XF} \ (\nu I^{+})$$

$$\frac{\begin{array}{l}\Gamma \vdash t : F[X := \nu XF] \\ \Gamma \vdash m : F\,\mathsf{mon}\,X\end{array}}{\Gamma \vdash \mathsf{out}^{-1}(m, t) : \nu XF} \ (\nu I^{i})$$

$$\frac{\Gamma \vdash r : \nu XF}{\Gamma \vdash \mathsf{out}\,r : F[X := \nu XF]} \ (\nu E)$$

- Operational semantics: given by the one-step β-reduction relation $t \to_\beta t'$ defined as the closure of the following axioms under all term formers.

$$\begin{array}{rcl}
\mathsf{It}(m,s,\mathsf{in}\,t) & \mapsto_\beta & s\Big(m\big(\lambda x.\mathsf{It}(m,s,x)\big)t\Big) \\
\mathsf{Rec}(m,s,\mathsf{in}\,t) & \mapsto_\beta & s\Big(m\Big(\langle \mathsf{Id}, \lambda z.\mathsf{Rec}(m,s,z)\rangle\Big)t\Big) \\
\mathsf{out}\,\mathsf{CoIt}(m,s,t) & \mapsto_\beta & m\big(\lambda z.\mathsf{CoIt}(m,s,z)\big)(st) \\
\mathsf{out}\,\mathsf{CoRec}(m,s,t) & \mapsto_\beta & m\big([\mathsf{Id}, \lambda x.\mathsf{CoRec}(m,s,x)]\big)(st) \\
\mathsf{out}\,\mathsf{out}^{-1}(m,t) & \mapsto_\beta & m(\lambda z z)t
\end{array}$$

  where $\mathsf{Id} := \lambda x.x$ and for given $f : A \to B, g : C \to B$ we define $[f,g] : A + C \to B$ as $[f,g] := \lambda z.\mathsf{case}(z, x.fx, y.gy)$. Analogously for $f : B \to A, g : B \to C$, $\langle f,g\rangle : B \to A \times C$ is defined as $\langle f,g\rangle := \lambda z.\langle fz, gz\rangle$.

  As usual the transitive closure of $\to_\beta$ is denoted by $\to_\beta^+$ and the reflexive-transitive closure by $\to_\beta^\star$.

These reduction rules correspond to the (co)iteration and (co)recursion principles obtained categorically in section 2.2, we can say that the rules are flat versions of the diagrams. The rule involving the out^{-1} constructor deserves some explanation, being out and out^{-1} inverses in the categorical setting it may seem strange not to define the rule directly as $\mathsf{out}\,\mathsf{out}^{-1}(m,t) \mapsto_\beta t$. This naive rule is ruled out as it destructs the termination of the system which can easily be seen as follows:
Define $\mathsf{T} := \nu X.X \to 1, m := \lambda f\lambda x\lambda y.\star$, $\omega := \lambda x.(\mathsf{out}\,x)x$, $\Omega := \omega(\mathsf{out}^{-1}(m,\omega))$. We have the typings $\vdash m : (X \to 1)\,\mathsf{mon}\,X$, $\vdash \omega : \mathsf{T} \to 1, \vdash \mathsf{out}^{-1}(m,\omega) : \mathsf{T}$ and $\vdash \Omega : 1$. With the rule $\mathsf{out}\,\mathsf{out}^{-1}(m,t) \mapsto_\beta t$ we get $\Omega \to_\beta^+ \Omega$:

$$\Omega \to_\beta (\mathsf{out}\,\mathsf{out}^{-1}(m,\omega))(\mathsf{out}^{-1}(m,\omega)) \to_\beta \omega(\mathsf{out}^{-1}(m,\omega)) \equiv \Omega$$

Therefore we rule out such reduction. This phenomen was originally noticed in [13] for fixed-point types.

The reader can notice that there is no rule corresponding to the inverse of the in morphism. Later we will discuss this ommision.

3.2. Programming in MICT

Given an inductive type μXF we can program functions

$$g : \mu XF \to B$$

using iteration or recursion as the operational semantics direct give us a nice reduction behavior.
If we define $g := \lambda z.\mathsf{It}(m,s,z)$ for given monotonicity witness m and step-function s the equation

$$g(\mathsf{in}\,x) = s\big(m(g)(x)\big)$$

holds, in the sense that $g(\mathsf{in}\,x) \to_\beta^+ s\big(m(g)x\big)$

Analogously primitive recursion provides a mean to program functions $g : \mu XF \to B$ which reduce as

$$g(\mathsf{in}\,x) \to_\beta^+ s\big(m(\langle \mathsf{Id}, g\rangle)(x)\big)$$

In this case g is defined as $g := \lambda z.\mathsf{Rec}(m,s,z)$.

In a dual way given a coinductive type νXF we can program functions

$$g : B \to \nu XF$$

by coiteration and corecursion as follows:

- If $g := \lambda x.\mathsf{CoIt}(m,s,x)$ then

$$\mathsf{out}\,g(x) \to_\beta^+ m(g)(s(x))$$

- If $g := \lambda x.\mathsf{CoRec}(m,s,x)$ then

$$\mathsf{out}\,g(x) \to_\beta^+ m([\mathsf{Id}, g])(s(x))$$

For an arbitrary inductive type μXF we denote the function $\lambda x.\,\mathsf{in}\,x$ simply with $\mathsf{in} : F[X := \mu XF] \to \mu XF$. Analogously $\mathsf{out} : \nu XF \to F[X := \nu XF]$ denotes the term $\lambda x.\,\mathsf{out}\,x$. The functions in and out play the role of encoded constructors and destructors respectively. This should be clear from the following examples.

Example 1 (The Unit Type)**.** *We will constantly need This type whose unique inhabitant is denoted by* $\star$ *and it is defined as* $1 := \forall X.X \to X$*. Later we will give a coinductive definition of* 1*. The unit type is mostly useful to define basic objects and to handle errors, in a sum type such as* $1 + A$ *it is useful to define an error constant* $\mathsf{error} := \mathsf{inl}\star$ *such that* $\mathsf{error} : 1 + A$*, this will be done in some of the examples below. The reader can observe that* $1 + A$ *is essentially the Haskell type* `maybe A`*.*

Example 2 (The natural numbers)**.** *Define* $\mathsf{nat} := \mu X.1 + X$ *with* $\mathsf{in} : 1 + \mathsf{nat} \to \mathsf{nat}$

- *Canonical monotonicity witness:* $\mathsf{map} := \lambda f\lambda x.\mathsf{case}(x, u.\,\mathsf{inl}\,u,\ v.\,\mathsf{inr}\,fv)$.
  In this and further examples we define so-called canonical witnesses which are automatically defined depending on the syntactical form of the type F*.*
- *Constructors:*
  - *Zero:* $0 : \mathsf{nat},\ 0 := \mathsf{in}(\mathsf{inl}\star)$
  - *Succesor function:* $\mathsf{suc} : \mathsf{nat} \to \mathsf{nat},\ \mathsf{suc} := \lambda n.\,\mathsf{in}(\mathsf{inr}\,n)$
- *Destructors:*
  - *Predeccesor function:* $\mathsf{pred} : \mathsf{nat} \to 1 + \mathsf{nat}$, *such that* $\mathsf{pred}\,0 = \mathsf{error},\ \mathsf{pred}(\mathsf{suc}\,n) = \mathsf{inr}\,n$
    $\mathsf{pred} := \lambda n.\mathsf{Rec}(\mathsf{map}, \lambda y.\mathsf{case}(y, u.\,\mathsf{inl}\,u, v.\,\mathsf{inr}(\mathsf{fst}\,v)), n)$
- *Some functions on* nat*:*
  - $\mathsf{sum} : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat},\ \ \mathsf{sum} := \lambda n\lambda z.\mathsf{It}(\mathsf{map}, [\lambda xn, \mathsf{suc}], z)$.
  - $\mathsf{prod} : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat},\ \ \mathsf{prod} := \lambda n\lambda z.\mathsf{It}(\mathsf{map}, [\lambda x.\mathsf{sum}\,x\,n], z)$.

Example 3 (Finite lists over A)**.** *Define* $\mathsf{list}(A) := \mu X.1 + A \times X$ *with* $\mathsf{in} : 1 + A \times \mathsf{list}(A) \to \mathsf{list}(A)$

- *Canonical monotonicity witness:* $\mathsf{map} := \lambda f\lambda x.\mathsf{case}(x, u.\,\mathsf{inl}\,u,\ v.\,\mathsf{inr}\langle\mathsf{fst}\,v, f\ \mathsf{snd}\,v\rangle)$
- *Constructors:*
  - *Empty list:* $\mathsf{nil} : \mathsf{list}(A),\quad \mathsf{nil} := \mathsf{in}(\mathsf{inl}\star)$
  - *Cons function:* $\mathsf{cons} : A \times \mathsf{list}(A) \to \mathsf{list}(A),\quad \mathsf{cons} := \lambda x.\,\mathsf{in}(\mathsf{inr}\,x)$
- *Destructors:*
  - *Head function:* $\mathsf{head} : \mathsf{list}(A) \to 1 + A$ *such that* $\mathsf{head}\,\mathsf{nil} = \mathsf{error},\ \mathsf{head}(\mathsf{cons}\langle a, \ell\rangle) = \mathsf{inr}\,a$
    $\mathsf{head} := \lambda z.\mathsf{Rec}(\mathsf{map}, \lambda y.\mathsf{case}(y, u.\,\mathsf{inl}\,u, v.\,\mathsf{inr}(\mathsf{fst}\,v)), z)$
  - *Tail function:* $\mathsf{tail} : \mathsf{list}(A) \to 1 + \mathsf{list}(A)$ *such that* $\mathsf{tail}\,\mathsf{nil} = \mathsf{error},\ \mathsf{tail}(\mathsf{cons}\langle a, \ell\rangle) = \mathsf{inr}\,\ell$
    $\mathsf{tail} := \lambda z.\mathsf{Rec}(\mathsf{map}, \lambda y.\mathsf{case}(y, u.\,\mathsf{inl}\,u, v.\,\mathsf{inr}(\mathsf{fst}(\mathsf{snd}\,v))), z)$
- *Some funcions on* $\mathsf{list}(A)$*:*
  - *Append:* $\mathsf{app} : \mathsf{list}(A) \to \mathsf{list}(A) \to \mathsf{list}(A)$,
    $\mathsf{app} := \lambda z.\mathsf{It}(\mathsf{map}, [\lambda y\lambda ww,\ \lambda p\lambda x, \mathsf{cons}\langle\mathsf{fst}\,p, (\mathsf{snd}\,p)x\rangle], z)$
  - *Length:* $\mathsf{length} : \mathsf{list}(A) \to \mathsf{nat},\ \mathsf{length} := \lambda z\mathsf{It}(\mathsf{map}, [\lambda x.0,\ \lambda p.\,\mathsf{suc}(\mathsf{snd}\,p)], z)$
  - *Reverse:* $\mathsf{rev} : \mathsf{list}(A) \to \mathsf{list}(A),\ \mathsf{rev} := \lambda z\mathsf{It}(\mathsf{map}, [\lambda x.\mathsf{nil},\ \lambda p.\,\mathsf{app}\,(\mathsf{snd}\,p)(\,\mathsf{cons}\langle\mathsf{fst}\,p, \mathsf{nil}\rangle)], z))$

Example 4 (Unlabelled binary trees)**.** *Define* $\mathsf{BinTree} := \mu X.1 + X \times X$ *with* $\mathsf{in} : 1 + \mathsf{BinTree} \times \mathsf{BinTree} \to \mathsf{BinTree}$

- *Canonical monotonicity witness:* $\mathsf{map} := \lambda f\lambda x.\mathsf{case}(x, u.\,\mathsf{inl}\,u, v.\,\mathsf{inr}\langle f(\mathsf{fst}\,v),\ f(\mathsf{snd}\,v)\rangle)$
- *Constructors:*
  - $\mathsf{node} : \mathsf{BinTree}\quad \mathsf{node} := \mathsf{in}(\mathsf{inl}\star)$
  - $\mathsf{makebt} : \mathsf{BinTree} \times \mathsf{BinTree} \to \mathsf{BinTree},\quad \mathsf{makebt} := \lambda x.\,\mathsf{in}(\mathsf{inr}\,x)$
- *Destructor:* $\mathsf{strees} : \mathsf{BinTree} \to 1 + \mathsf{BinTree}$ *such that* $\mathsf{strees}\,\mathsf{node} = \mathsf{error},\ \ \mathsf{strees}\,\mathsf{makebt}\,\langle t_1, t_2\rangle = \mathsf{inr}\langle t_1, t_2\rangle$
  $\mathsf{strees} := \lambda x.\mathsf{Rec}(\mathsf{map}, \lambda y.\mathsf{case}(y, u.\,\mathsf{inl}\,u, v.\,\mathsf{inr}\langle\mathsf{fst}(\mathsf{fst}\,v), \mathsf{fst}(\mathsf{snd}\,v)\rangle), x)$
- *Some functions on* $\mathsf{BinTree}$
  - $\mathsf{nn} : \mathsf{BinTree} \to \mathsf{nat}$ *returns the number of nodes given by*

    $$\mathsf{nn}\,\mathsf{node} = 1,\ \mathsf{nn}(\mathsf{makebt}\langle t_1, t_2\rangle) = \mathsf{suc}(\mathsf{sum}\,(\mathsf{nn}\,t_1)\,(\mathsf{nn}\,t_2)).$$

    $$\mathsf{nn} := \lambda x.\mathsf{It}(\mathsf{map}, \lambda y.\mathsf{case}(y, u.1, v.\,\mathsf{suc}(\mathsf{sum}(\mathsf{fst}\,v)(\mathsf{snd}\,v)), x)$$

Example 5 (Unlabelled A-branching well-founded trees with succesor)**.** $\mathsf{WFTreeS}\,A := \mu X.1 + X + (A \to X)$ *with constructors*

- $\mathsf{node} : \mathsf{WFTreeS}\, A, \quad \mathsf{node} := \mathsf{in}(\mathsf{inl}\, \star)$
- $\mathsf{succ} : \mathsf{WFTreeS}\, A \to \mathsf{WFTreeS}\, A \quad \mathsf{succ} := \lambda x.\, \mathsf{in}(\mathsf{inr}(\mathsf{inl}\, x))$
- $\mathsf{makewft} : (A \to \mathsf{WFTreeS}\, A) \to \mathsf{WFTreeS}\, A \quad \mathsf{makewft} := \lambda f.\, \mathsf{in}(\mathsf{inr}(\mathsf{inr}\, f))$

The particular instance $\mathcal{O} := \mathsf{WFTreeS}\, \mathsf{nat}$ rises the type of Brouwer ordinals or Kleene's $\mathcal{O}$.

Example 6 (Streams (infinite lists) over A)**.** $\mathsf{stream}(A) := \nu X.A \times X$ *with* $\mathsf{out} : \mathsf{stream}(A) \to A \times \mathsf{stream}(A)$

- *Canonical monotonicity witness:* $\mathsf{map} := \lambda f \lambda x.\langle \mathsf{fst}\, x, f\, \mathsf{snd}\, x\rangle$
- *Destructors:*
  - $\mathsf{head} : \mathsf{stream}(A) \to A, \quad \mathsf{head} := \lambda x.\, \mathsf{fst}(\mathsf{out}\, x)$
  - $\mathsf{tail} : \mathsf{stream}(A) \to \mathsf{stream}(A), \quad \mathsf{tail} := \lambda x.\, \mathsf{snd}(\mathsf{out}\, x)$
- *Constructor* $\mathsf{cons} : A \times \mathsf{stream}(A) \to \mathsf{stream}(A)$
- *Some functions:*
  - $\mathsf{cnt} : A \to \mathsf{stream}(A)$ *giving a stream of constants*
    $\mathsf{head}(\mathsf{cnt}\, a) = a, \ \mathsf{tail}(\mathsf{cnt}\, a) = \mathsf{cnt}\, a$
    $\mathsf{cnt} := \lambda z \mathsf{Colt}(\mathsf{map}, \lambda y.\langle y, y\rangle, z)$
  - $\mathsf{from} : \mathsf{nat} \to \mathsf{stream}(\mathsf{nat})$ *returning the stream of naturals from the given one*
    $\mathsf{head}(\mathsf{from}\, n) = n, \ \mathsf{tail}(\mathsf{from}\, n) = \mathsf{from}(\mathsf{suc}\, n)$
    $\mathsf{from} := \lambda z \mathsf{Colt}(\mathsf{map}, \lambda y.\langle y, \mathsf{suc}\, y\rangle, z)$
  - $\mathsf{mapstr} : (A \to B) \times \mathsf{stream}(A) \to \mathsf{stream}(B)$ *returning the stream resulting of applying f to the elements in a given stream.*
    $\mathsf{head}(\mathsf{mapstr}\langle f, s\rangle) = f\, (\mathsf{head}\, s), \ \mathsf{tail}(\mathsf{mapstr}\langle f, s\rangle) = \mathsf{mapstr}\langle f, \mathsf{tail}\, s\rangle$
    $\mathsf{mapstr} := \lambda z \mathsf{Colt}(\mathsf{map}, \lambda p.\langle (\mathsf{fst}\, p)(\mathsf{head}(\mathsf{snd}\, p)), \langle \mathsf{fst}\, p, \ \mathsf{tail}(\mathsf{snd}\, p)\rangle\rangle, z)$

Of course it would be more elegant to give a curried version of mapstr*. From now on, whenever possible, we will give only curried functions.*

Example 7 (Conatural numbers (the ordinal $\omega + 1$))**.** $\mathsf{conat} := \nu X.1 + X$

- *Canonical monotonicity witness:* $\mathsf{map} := \lambda f \lambda x.\mathsf{case}(x, u.\, \mathsf{inl}\, \star, v.\, \mathsf{inr}(f\, v))$
- *Destructor:* $\mathsf{pred} : \mathsf{conat} \to 1 + \mathsf{conat}, \quad \mathsf{pred} := \mathsf{out}.$
  Observe that this is the predeccesor function with error such that $\mathsf{pred}\, 0 = \mathsf{error}$
- *Constructors:*
  - *Zero* $0 : \mathsf{conat}, \ 0 := \mathsf{out}^{-1}(\mathsf{map}, \mathsf{inl}\, \star)$
  - *Succesor* $\mathsf{suc} : \mathsf{conat} \to \mathsf{conat}, \ \mathsf{suc} := \lambda n.\, \mathsf{out}^{-1}(\mathsf{map}, \mathsf{inr}\, n)$
  - *Omega: to define the ordinal ω we first define a global element of* $\mathsf{conat}, \ \omega^\dagger : 1 \to \mathsf{conat}$ *by coiteration as:* $\omega^\dagger := \lambda x.\mathsf{Colt}(\mathsf{map}, \mathsf{inr}, x)$*. Omega is then defined as* $\omega := \omega^\dagger \star$*. With this definition we have* $\mathsf{pred}(\mathsf{suc}\, \omega) \to^\star \mathsf{inr}\, \omega$*.*
- *Some functions on* conat*:*
  - *Sum of conaturals: the sum $\oplus$ is destructed as follows:*

$$\mathsf{pred}(0 \oplus 0) = \mathsf{error} \quad \mathsf{pred}(0 \oplus (\mathsf{suc}\, m)) = 0 \oplus m \quad \mathsf{pred}((\mathsf{suc}\, n) \oplus m) = n \oplus m$$

We program an uncurried version $\oplus : \mathsf{conat} \times \mathsf{conat} \to \mathsf{conat}$ *as* $\oplus := \lambda p.\mathsf{Colt}(\mathsf{map}, s, p)$*, where the step function* $s : \mathsf{conat} \times \mathsf{conat} \to 1 + \mathsf{conat} \times \mathsf{conat}$ *is such that:*

$$s\ \langle x, y\rangle \quad = \quad \begin{cases} \mathsf{error} & if\ \mathsf{pred}\, x = \mathsf{pred}\, y = \mathsf{error} \\ \mathsf{inr}\, \langle x', y\rangle & if\ \mathsf{pred}\, x = \mathsf{inr}\, x' \\ \mathsf{inr}\, \langle x, y'\rangle & if\ \mathsf{pred}\, x = \mathsf{error}, \quad \mathsf{pred}\, y = \mathsf{inr}\, y' \end{cases}$$

  - *Product of conaturals: such that $n \otimes m$ is the usual product of naturals and $n \otimes \omega = \omega \otimes n = \omega$. A definition analogous to the sum above is obtained by coiteration with the following step function:*

$$s\ \langle x, y\rangle \quad = \quad \begin{cases} \mathsf{error} & if\ \mathsf{pred}\, x = \mathsf{error} \\ \mathsf{inr}\, \langle x', y\rangle & if\ \mathsf{pred}\, x = \mathsf{inr}\, x' \\ \mathsf{inr}\, \langle x, y'\rangle & if\ \mathsf{pred}\, x = \mathsf{inr}\, x', \ \mathsf{pred}\, x' = \mathsf{error}, \quad \mathsf{pred}\, y = \mathsf{inr}\, y' \end{cases}$$

Example 8 (Finite and infinite lists over A)**.** $\mathsf{FinInfList}\, A := \nu X.1 + A \times X$ *with destructor*

- $\mathsf{ht} : \mathsf{FinInfList}\, \mathsf{A} \to 1 + \mathsf{A} \times \mathsf{FinInfList}\, \mathsf{A}, \quad \mathsf{ht} := \mathsf{out}$

Observe that with this definition it is inelegant to define the usual destructors head *and* tail*, for we need a case analysis. This problem can be avoided if we define instead the type as* $\mathsf{FinInfList}\, A := \nu X.(1 + A) \times (1 + X)$*. This definition yields an easy definition of destructors:* $\mathsf{head} := \mathsf{fst} \circ \mathsf{out}$, $\mathsf{tail} := \mathsf{snd} \circ \mathsf{out}$*. However we still need to work with projections.*

From all the above examples we can observe the heavy use of injections and projections which complicate both the definitions of map witnesses and of functions in general, for the constructors or destructors of a type are encoded and not available directly. After the proof of strong normalization of this system we will present an improved system which allows to express directly the constructors and destructors, and therefore modularizes the definition of types and functions.

3.3. A Word on Positivity

The reader can confirm that the above examples are in fact positive, that is, the variable X occurs only on positive positions in the type F. An immediate question arises: what are then the advantages of using full monotonicity ?

Above all, positive systems are incompatible with Curry-style in the following sense: In a positive system a functor is a pair $\langle \lambda XF, \mathsf{map}_{\lambda XF} \rangle$ where X occurs only positive in F and the monotonicity witness is fixed and defined according to the shape of F, esentially by the rules given in appendix B. Observe that this term is annotated by the type expression λXF.The (co)inductive types are only allowed if the positivity restriction holds, therefore there is no neccesity to attach the witness in the typing rules. For example the (μE) rule becomes

$$\frac{\Gamma \vdash t : \mu XF \qquad \Gamma \vdash s : F[X := B] \to B}{\Gamma \vdash \mathsf{It}(s, t) : B}$$

and the corresponding reduction rule becomes:

$$\mathsf{It}(s, \mathsf{in}\, t) \mapsto_\beta s\Big(\mathsf{map}\big(\lambda x.\mathsf{It}(s, x)\big)t\Big)$$

But there is no way to recover the specific term map only from the terms. It is mandatory either to look for the adequate instance of (μE) or to annotate some term to recover the map term. In the first case we would get a conditional term rewrite system very hard to handle from the metatheorical point of view. In the second case we would obviously get a Church-style system, and the rule becomes, for instance

$$\mathsf{It}_{\mu XF}(s, \mathsf{in}\, t) \mapsto_\beta s\Big(\mathsf{map}_{\lambda XF}\big(\lambda x.\mathsf{It}(s, x)\big)t\Big)$$

Other answers in favor of monotonicity are:

- Specific monotonicity witnesses are not involved in proofs, we can even have hypothetical monotonicity, i.e. just an additional assumption $x : F\, \mathsf{mon}\, X$ in our context. Therefore the generality of our approach simplifies proofs.
- For higher-order systems there is no fixed concept of positivity. With full monotonicity we can generalize directly the systems presented in this work, this has been done in [14], [1]. Moreover, sometimes different witnesses are useful for programming, see the example on power list reverse in [1].

3.4. MICT is safe

We prove the safety of our system by proving termination and type-preservation.

3.4.1. *Strong Normalization*

In this section we show termination of the type system by showing directly that every typable term in MICT strongly normalizes by means of a variation of the well-known Tait's method using so-called saturated sets. This method characterizes the typable strongly normalizing terms in a syntax-directed way and modularizes in a convenient way the normalization proof. The proof presented here, specifically

the constructions on saturated sets, is based in proofs given in [11] for related inductive type systems in Church-style. The coinductive constructions are, to our knowledge, new.

Definition 6. *A term t is strongly normalizing with respect to a reduction relation $\rightarrow$ if there is no infinite reduction sequence $t \rightarrow t_1 \rightarrow t_2 \rightarrow \ldots$. Equivalently, the set* sn *of strongly normalizing terms can be defined inductively as follows:*

$$\frac{\textit{For all } t', t \rightarrow t' \textit{ implies } t' \in \mathsf{sn}}{t \in \mathsf{sn}}$$

The syntactical concept of elimination is central to our proof.

Definition 7. *Let $\star$ be a new symbol. An elimination is an expression of one of the following forms:*

$$\star s,\ \mathsf{case}(\star, x.t, y.r),\ \mathsf{fst}\,\star,\ \mathsf{snd}\,\star,\ \mathsf{It}(m, s, \star),\ \mathsf{Rec}(m, s, \star),\ \mathsf{out}\,\star$$

eliminations are denoted with the letter e.

Definition 8. *Multiple eliminations are defined as follows:*

$$E ::= \star \mid e[\star := E]$$

where the substitution $e[\star := E]$ is defined as if $\star$ were a term variable. From now on we will use $E[r]$ to denote $E[\star := r]$.

Definition 9. *The set* SN *is inductively defined as follows:*

$$\frac{}{x \in \mathsf{SN}} \qquad \frac{E[x], s \in \mathsf{SN}}{E[x]s \in \mathsf{SN}} \qquad \frac{E[x], s, t \in \mathsf{SN}}{\mathsf{case}(E[x], x.s, y.t) \in \mathsf{SN}}$$

$$\frac{E[x] \in \mathsf{SN}}{\mathsf{fst}(E[x]) \in \mathsf{SN}} \qquad \frac{E[x] \in \mathsf{SN}}{\mathsf{snd}(E[x]) \in \mathsf{SN}}$$

$$\frac{r \in \mathsf{SN}}{\lambda x r \in \mathsf{SN}} \qquad \frac{E\big[r[x := s]\big], s \in \mathsf{SN}}{E[(\lambda x r)s] \in \mathsf{SN}} \qquad \frac{t \in \mathsf{SN}}{\mathsf{inl}\, t \in \mathsf{SN}} \qquad \frac{t \in \mathsf{SN}}{\mathsf{inr}\, t \in \mathsf{SN}}$$

$$\frac{E\big[r[x := t]\big], s \in \mathsf{SN}}{E[\mathsf{case}(\mathsf{inl}\, t, x.r, y.s)] \in \mathsf{SN}} \qquad \frac{E\big[s[y := t]\big], r \in \mathsf{SN}}{E\big[\mathsf{case}(\mathsf{inr}\, t, x.r, y.s)\big] \in \mathsf{SN}}$$

$$\frac{r, s \in \mathsf{SN}}{\langle r, s\rangle \in \mathsf{SN}} \qquad \frac{E[r], s \in \mathsf{SN}}{E[\mathsf{fst}\langle r, s\rangle] \in \mathsf{SN}} \qquad \frac{E[s], r \in \mathsf{SN}}{E[\mathsf{snd}\langle r, s\rangle] \in \mathsf{SN}}$$

$$\frac{m, s, E\big[x\big] \in \mathsf{SN}}{\mathsf{It}(m, s, E\big[x\big]) \in \mathsf{SN}} \qquad \frac{m, s, E\big[x\big] \in \mathsf{SN}}{\mathsf{Rec}(m, s, E\big[x\big]) \in \mathsf{SN}}$$

$$\frac{E\big[x\big] \in \mathsf{SN}}{\mathsf{out}\, E\big[x\big] \in \mathsf{SN}}$$

$$\frac{t \in \mathsf{SN}}{\mathsf{in}\, t \in \mathsf{SN}} \qquad \frac{E\big[s\Big(m\big(\lambda x.\mathsf{It}(m, s, x)\big)t\Big)\big] \in \mathsf{SN}}{E\big[\mathsf{It}(m, s, \mathsf{in}\, t)\big] \in \mathsf{SN}}$$

$$\frac{E\big[s\Big(m\big(\langle \mathsf{Id}, \lambda x.\mathsf{Rec}(m, s, x)\rangle\big)t\Big)\big] \in \mathsf{SN}}{E\big[\mathsf{Rec}(m, s, \mathsf{in}\, t)\big] \in \mathsf{SN}}$$

$$\frac{m, s, t \in \mathsf{SN}}{\mathsf{CoIt}(m, s, t) \in \mathsf{SN}} \qquad \frac{m, s, t \in \mathsf{SN}}{\mathsf{CoRec}(m, s, t) \in \mathsf{SN}} \qquad \frac{m, t \in \mathsf{SN}}{\mathsf{out}^{-1}(m, t) \in \mathsf{SN}}$$

$$\frac{E\big[m\big(\lambda z.\mathsf{CoIt}(m, s, z)\big)(st)\big] \in \mathsf{SN}}{E\big[\,\mathsf{out}\,\mathsf{CoIt}(m, s, t)\big] \in \mathsf{SN}} \qquad \frac{E\big[m\big([\mathsf{Id}, \lambda z.\mathsf{CoRec}(m, s, z)]\big)(st)\big] \in \mathsf{SN}}{E\big[\,\mathsf{out}\,\mathsf{CoRec}(m, s, t)\big] \in \mathsf{SN}}$$

$$\frac{E\big[m(\lambda z z)t\big] \in \mathsf{SN}}{E\big[\,\mathsf{out}\,\mathsf{out}^{-1}(m, t)\big] \in \mathsf{SN}}$$

This definition of SN captures exhaustively the closure of terms under reduction without refering to the reduction relation itself. Moreover it is based only on the typing rules, each expression (term or elimination) on the conclusion of a rule can only be derived with one of the typing rules.

Proposition 3. *The defining rules of* SN *are sound with respect to the reduction relation. That is,* $\mathsf{SN} \subseteq \mathsf{sn}$

Proof. Several routinary inductions show that sn is closed under all the defining rules of SN. therefore the claim follows by minimality of SN. □

Next we define saturated sets after the set SN. These sets are needed to recursively define the so-called predicates of strong computability starting from a candidate assigment, which is an assignment of saturated sets for type variables.

Definition 10 (Saturated Set)**.** *A set* $\mathcal{M}$ *of terms is saturated if and only if:*

$$\mathcal{M} \subseteq \mathsf{SN}$$

and if the following closure conditions hold:

$$\frac{E[x] \in \mathsf{SN}}{E[x] \in \mathcal{M}}$$

$$\frac{E\big[r[x := s]\big] \in \mathcal{M} \quad s \in \mathsf{SN}}{E[(\lambda x r)s] \in \mathcal{M}}$$

$$\frac{E\big[r[x := t]\big] \in \mathcal{M} \quad s \in \mathsf{SN}}{E[\mathsf{case}(\mathsf{inl}\, t, x.r, y.s)] \in \mathcal{M}} \qquad \frac{E\big[s[y := t]\big] \in \mathcal{M} \quad r \in \mathsf{SN}}{E\big[\mathsf{case}(\mathsf{inr}\, t, x.r, y.s)\big] \in \mathcal{M}}$$

$$\frac{E[r] \in \mathcal{M} \quad s \in \mathsf{SN}}{E[\mathsf{fst}\langle r, s\rangle] \in \mathcal{M}} \qquad \frac{E[s] \in \mathcal{M} \quad r \in \mathsf{SN}}{E[\mathsf{snd}\langle r, s\rangle] \in \mathcal{M}}$$

$$\frac{E\big[s\Big(m\big(\lambda x.\mathsf{It}(m, s, x)\big)t\Big)\big] \in \mathcal{M}}{E\big[\mathsf{It}(m, s, \mathsf{in}\, t)\big] \in \mathcal{M}}$$

$$\frac{E\big[s\Big(m\big(\langle \mathsf{Id}, \lambda x.\mathsf{Rec}(m, s, x)\rangle\big)t\Big)\big] \in \mathcal{M}}{E\big[\mathsf{Rec}(m, s, \mathsf{in}\, t)\big] \in \mathcal{M}}$$

$$\frac{E\big[m\big(\lambda z.\mathsf{CoIt}(m, s, z)\big)(st)\big] \in \mathcal{M}}{E\big[\,\mathsf{out}\,\mathsf{CoIt}(m, s, t)\big] \in \mathcal{M}} \qquad \frac{E\big[m\big([\mathsf{Id}, \lambda z.\mathsf{CoRec}(m, s, z)]\big)(st)\big] \in \mathcal{M}}{E\big[\,\mathsf{out}\,\mathsf{CoRec}(m, s, t)\big] \in \mathcal{M}}$$

$$\frac{E\big[m(\lambda z z)t\big] \in \mathcal{M}}{E\big[\,\mathsf{out}\,\mathsf{out}^{-1}(m, t)\big] \in \mathcal{M}}$$

the set of saturated sets will be denoted with SAT*.*

Proposition 4. $\mathsf{SN} \in \mathsf{SAT}$ *and* SAT *is closed under intersection.*

Proof. Straightforward □

Definition 11. *Given a set of terms* M *we define the saturated closure of* M *as follows:*

$$\mathsf{cl}(M) := \bigcap\{\mathcal{N} \in \mathsf{SAT} \mid M \cap \mathsf{SN} \subseteq \mathcal{N}\}$$

$\mathsf{cl}(M)$ is the least saturated superset of $M \cap \mathsf{SN}$. Observe that $M \subseteq \mathsf{cl}(M)$ if and only if $M \subseteq \mathsf{SN}$.

Central to the proof of strong normalization are the constructions for saturated sets corresponding to the type constructors of the system. In order for the proof to work these constructions must be sound with respect to the typing rules of the system. We define the constructions and prove the soundness in the next sections.

3.4.2. *Saturated Sets for Function, Sum and Product Types*

Definition 12. *Given a variable x and $\mathcal{M},\mathcal{N} \in \mathsf{SAT}$ we define*

$$\mathsf{S}_x(\mathcal{M},\mathcal{N}) := \{t \mid \forall s \in \mathcal{M}.\, t[x := s] \in \mathcal{N}\}$$

Definition 13. *Given $\mathcal{M},\mathcal{N} \in \mathsf{SAT}$, we define the following sets:*

$$\begin{aligned}
\mathcal{M} \to \mathcal{N} &:= \mathsf{cl}(\{\lambda x t \mid t \in \mathsf{S}_x(\mathcal{M},\mathcal{N})\}) \\
\mathcal{M} + \mathcal{N} &:= \mathsf{cl}(\{\mathsf{inl}\, t \mid t \in \mathcal{M}\} \cup \{\mathsf{inr}\, t \mid t \in \mathcal{N}\}) \\
\mathcal{M} \times \mathcal{N} &:= \mathsf{cl}(\{\langle s,t\rangle \mid s \in \mathcal{M} \text{ and } t \in \mathcal{N}\})
\end{aligned}$$

Obviously these are saturated sets.

Proposition 5 (Soundness of the constructions)**.** *Assume $\mathcal{M},\mathcal{N} \in \mathsf{SAT}$, then*

(1) *If $s \in \mathcal{M}$ and $t \in \mathcal{N}$ then $\langle s,t\rangle \in \mathcal{M} \times \mathcal{N}$*
(2) *If $r \in \mathcal{M} \times \mathcal{N}$ then $\mathsf{fst}\, r \in \mathcal{M}$ and $\mathsf{snd}\, r \in \mathcal{N}$*
(3) *If $t \in \mathcal{M}$ then $\mathsf{inl}\, t \in \mathcal{M} + \mathcal{N}$.*
(4) *If $t \in \mathcal{N}$ then $\mathsf{inr}\, t \in \mathcal{M} + \mathcal{N}$.*
(5) *If $r \in \mathcal{M} + \mathcal{N}$, $s \in \mathsf{S}_x(\mathcal{M},\mathcal{P})$, $t \in \mathsf{S}_y(\mathcal{N},\mathcal{P})$ then $\mathsf{case}(r, x.s, y.t) \in \mathcal{P}$*
(6) *If $t \in \mathsf{S}_x(\mathcal{M},\mathcal{N})$ then $\lambda x t \in \mathcal{M} \to \mathcal{N}$.*
(7) *If $r \in \mathcal{M} \to \mathcal{N}$ and $s \in \mathcal{M}$ then $rs \in \mathcal{N}$.*

Proof. For part 1 set $\mathcal{I}_\times(\mathcal{M},\mathcal{N}) = \{\langle s,t\rangle \mid s \in \mathcal{M} \text{ and } t \in \mathcal{N}\}$. Clearly $\mathcal{I}_\times(\mathcal{M},\mathcal{N}) \subseteq \mathsf{SN}$ and therefore $\mathcal{I}_\times(\mathcal{M},\mathcal{N}) \subseteq \mathcal{M} \times \mathcal{N}$. The claim is now obvious. For part 2 set $\mathcal{E}_\times(\mathcal{M},\mathcal{N}) = \{r \in \mathsf{SN} \mid \mathsf{fst}\, r \in \mathcal{M} \text{ and } \mathsf{snd}\, r \in \mathcal{N}\} \subseteq \mathsf{SN}$. We have to show that $\mathcal{E}_\times(\mathcal{M},\mathcal{N})$ is closed under the rules for saturated sets. As example take $E\big[[r[x := s]\big] \in \mathcal{E}_\times(\mathcal{M},\mathcal{N})$ and $s \in \mathsf{SN}$. We have to show $E[(\lambda x r)s] \in \mathcal{E}_\times(\mathcal{M},\mathcal{N})$. $E\big[[r[x := s]\big] \in \mathcal{E}_\times(\mathcal{M},\mathcal{N})$ implies $\mathsf{fst}(E[\big[r[x := s]\big]) \in \mathcal{M}$ and $\mathsf{snd}(E[\big[r[x := s]\big]) \in \mathcal{N}$. Observe that $\mathsf{fst}(E\big[r[x := s]\big]) \equiv (\mathsf{fst}\star)\big[\star := E\big[r[x := s]\big]\big] \equiv (\mathsf{fst}\star)[\star := E\big[r[x := s]\big]$ and that $(\mathsf{fst}\star)[\star := E]$ is again a multiple elimination say E', therefore we have $E'\big[r[x := s]\big] \in \mathcal{M}$, and as $s \in \mathsf{SN}$ and $\mathcal{M} \in \mathsf{SAT}$ we get $E'[(\lambda x r)s] \in \mathsf{SN}$, i.e., $\mathsf{fst}(E[(\lambda x r)s]) \in \mathcal{M}$. Analogously we show that $\mathsf{snd}(E[(\lambda x r)s]) \in \mathcal{N}$. Therefore $E[(\lambda x r)s] \in \mathcal{E}_\times(\mathcal{M},\mathcal{N})$. The other rules for saturated sets are proved similarly.
The remaining claims are proved analogously, here we only give the needed sets: for parts $3, 4$, $\mathcal{I}_+(\mathcal{M},\mathcal{N}) = \{\mathsf{inl}\, t \mid t \in \mathcal{M}\} \cup \{\mathsf{inr}\, t \mid t \in \mathcal{N}\}$. For part 5 use $\mathcal{E}_+(\mathcal{M},\mathcal{N}) = \{r \in \mathsf{SN} \mid \forall \mathcal{P}\, \forall x\, \forall s \in \mathsf{S}_x(\mathcal{M},\mathcal{P})\, \forall y\, \forall t \in \mathsf{S}_y(\mathcal{N},\mathcal{P}).\, \mathsf{case}(r, x.s, y.t) \in \mathcal{P}\}$. For the claims concerning $\mathcal{M} \to \mathcal{N}$ take $\mathcal{I}_\to(\mathcal{M},\mathcal{N}) := \{\lambda x t \mid t \in \mathsf{S}_x(\mathcal{M},\mathcal{N})\}$ and $\mathcal{E}_\to(\mathcal{M},\mathcal{N}) := \{r \in \mathsf{SN} \mid \forall s \in \mathcal{M}.\, rs \in \mathcal{N}\}$ □

3.4.3. *Saturated Sets for Inductive Types*

The construction for inductive types are based on the ones given in [11] for related systems in Church-style. Related constructions for arbitrary fixed points can be found in [15].

From now on, we fix $\Phi : \mathsf{SAT} \to \mathsf{SAT}$.

Definition 14. *Given $\mathcal{M} \in \mathsf{SAT}$ we define $\mathcal{I}_\mu(\mathcal{M}) := \{\mathsf{in}\, r \mid r \in \Phi(\mathcal{M})\}$ and $\Psi_I : \mathsf{SAT} \to \mathsf{SAT}$ as $\Psi_I(\mathcal{M}) := \mathsf{cl}(\mathcal{I}_\mu(\mathcal{M}))$.*

As we do not assume that Φ is monotone, we cannot prove neither that Ψ_I is monotone. This is an essential difference with the treatment in definition 27, page 221 in [15]. We proceed as follows. Set

$$\mathsf{mon}(\Phi) := \bigcap_{\mathcal{P},\mathcal{Q} \in \mathsf{SAT}} (\mathcal{P} \to \mathcal{Q}) \to (\Phi(\mathcal{P}) \to \Phi(\mathcal{Q}))$$

and define $\Phi^{\supseteq} : \mathsf{SAT} \to \mathcal{P}(\mathsf{SN})$ as:

$$\Phi^{\supseteq}(\mathcal{M}) := \{t \in \mathsf{SN} \mid \forall m \in \mathsf{mon}(\Phi), \forall \mathcal{N} \in \mathsf{SAT}, \forall s \in \mathcal{M} \to \mathcal{N}.mst \in \Phi(\mathcal{N})\}$$

Lemma 1. *For all $\mathcal{P},\mathcal{Q},\mathcal{N} \in \mathsf{SAT}$. If $\mathcal{P} \subseteq \mathcal{Q}$ then $\mathcal{Q} \to \mathcal{N} \subseteq \mathcal{P} \to \mathcal{N}$.*

Proof. Assume $\mathcal{P} \subseteq \mathcal{Q}$. It suffices to show $\mathcal{I}_{\to}(\mathcal{Q},\mathcal{N}) \cap \mathsf{SN} = \mathcal{I}_{\to}(\mathcal{Q},\mathcal{N}) \subseteq \mathcal{P} \to \mathcal{N}$. Take $\lambda xt \in \mathcal{I}_{\to}(\mathcal{Q},\mathcal{N})$, i.e., $t \in \mathsf{S}_x(\mathcal{Q},\mathcal{N})$. To show $\lambda xt \in \mathcal{P} \to \mathcal{N}$ it suffices to prove $t \in \mathsf{S}_x(\mathcal{P},\mathcal{N})$. Therefore we take $p \in \mathcal{P}$ and show $t[x := p] \in \mathcal{N}$, but this is clear from $t \in \mathsf{S}_x(\mathcal{Q},\mathcal{N})$ because by assumption we also have $p \in \mathcal{Q}$. □

Corollary 1. *$\Phi^{\supseteq}$ is monotone, i.e., for all $\mathcal{P}, \mathcal{Q}, \in \mathsf{SAT}$, if $\mathcal{P} \subseteq \mathcal{Q}$ then $\Phi^{\supseteq}(\mathcal{P}) \subseteq \Phi^{\supseteq}(\mathcal{Q})$.*

Proof. Assume $\mathcal{P} \subseteq \mathcal{Q}$ and take $t \in \Phi^{\supseteq}(\mathcal{P})$. Take also $\mathcal{N} \in \mathsf{SAT}$, $m \in \mathsf{mon}(\Phi)$ and $s \in \mathcal{Q} \to \mathcal{N}$. We need to show $mst \in \Phi(\mathcal{N})$. By the previous lemma $s \in \mathcal{Q} \to \mathcal{N}$ implies $s \in \mathcal{P} \to \mathcal{N}$. The claim follows now from the assumption $t \in \Phi^{\supseteq}(\mathcal{P})$.

□

Next define $\mathcal{I}^{\supseteq}_{\mu}(\mathcal{M}) := \{\mathsf{in}\, r \mid r \in \Phi^{\supseteq}(\mathcal{M})\}$ and $\Psi^{\supseteq}_{I} : \mathsf{SAT} \to \mathsf{SAT}$ as $\Psi^{\supseteq}_{I}(\mathcal{M}) := \mathsf{cl}(\mathcal{I}^{\supseteq}_{\mu}(\mathcal{M}))$
Clearly $\Psi^{\supseteq}_{I}$ is monotone, because so is $\Phi^{\supseteq}$, therefore the following definition is correct

$$\mu(\Phi) := \mathsf{lfp}(\Psi^{\supseteq}_{I}).$$

i.e. $\mu(\Phi)$ is the least fixed point of $\Psi^{\supseteq}_{I}$.

Lemma 2. *$\mathcal{I}_{\mu}(\mathcal{M}) \subseteq \mathsf{SN}$ and $\mathcal{I}^{\supseteq}_{\mu}(\mathcal{M}) \subseteq \mathsf{SN}$.*

Proof. We show the second claim. Take $t \in \mathcal{I}^{\supseteq}_{\mu}(\mathcal{M})$, that is, $t \equiv \mathsf{in}\, r$ with $r \in \Phi^{\supseteq}(\mathcal{M})$. As $\Phi^{\supseteq}(\mathcal{M}) \subseteq \mathsf{SN}$ we have $r \in \mathsf{SN}$, which by definition of SN implies $\mathsf{in}\, r \in \mathsf{SN}$, i.e., $t \in \mathsf{SN}$.

Corollary 2. *$\mathcal{I}_{\mu}(\mathcal{M}) \subseteq \Psi_{I}(\mathcal{M})$ and $\mathcal{I}^{\supseteq}_{\mu}(\mathcal{M}) \subseteq \Psi^{\supseteq}_{I}(\mathcal{M})$.*

Proof. We proof the second claim. By definition of the closure we have $\mathcal{I}^{\supseteq}_{\mu}(\mathcal{M}) \cap \mathsf{SN} \subseteq \Psi^{\supseteq}_{I}(\mathcal{M})$. But the previous lemma yields $\mathcal{I}^{\supseteq}_{\mu}(\mathcal{M}) \cap \mathsf{SN} = \mathcal{I}^{\supseteq}_{\mu}(\mathcal{M})$. □

Definition 15. *Given $\Phi : \mathsf{SAT} \to \mathsf{SAT}$ and $\mathcal{M} \in \mathsf{SAT}$ we define*

$$\mathcal{E}_{\mu}(\mathcal{M}) := \Big\{ r \in \mathsf{SN} \;\Big|\; \begin{array}{l} \forall m \in \mathsf{mon}(\Phi).\, \forall \mathcal{N} \in \mathsf{SAT}. \\ \big(\forall s \in \Phi(\mathcal{N}) \to \mathcal{N}.\, \mathsf{It}(m,s,r) \in \mathcal{N}\big) \wedge \\ \big(\forall s \in \Phi(\mathcal{M} \times \mathcal{N}) \to \mathcal{N}.\, \mathsf{Rec}(m,s,r) \in \mathcal{N}\big)\} \end{array}$$

and $\Psi_{E} : \mathsf{SAT} \to \mathsf{SAT}$ as

$$\Psi_{E}(\mathcal{M}) := \mathsf{cl}(\mathcal{E}_{\mu}(\mathcal{M})).$$

Lemma 3. *$\mathcal{E}_{\mu}(\mathcal{M}) \in \mathsf{SAT}$.*

Proof. Is clear that $\mathcal{E}_{\mu}(\mathcal{M}) \subseteq \mathsf{SN}$.
Take $E[x] \in \mathsf{SN}$. We have to show that $E[x] \in \mathcal{E}_{\mu}(\mathcal{M})$. Fix $m \in \mathsf{mon}(\Phi), \mathcal{N} \in \mathsf{SAT}$.

- Assume $s \in \Phi(\mathcal{N}) \to \mathcal{N}$.
  The goal is $\mathsf{It}(m,s,E[x]) \in \mathcal{N}$. Observe that this term is again a multiple elimination say $E'[x]$. As $\mathcal{N} \in \mathsf{SAT}$ it suffices to show that $E'[x] \in \mathsf{SN}$. We have $E[x] \in \mathsf{SN}$ and $s \in \Phi(\mathcal{N}) \to \mathcal{N} \subseteq \mathsf{SN}$ implies $s \in \mathsf{SN}$, similarly $m \in \mathsf{mon}(\Phi) \subseteq \mathsf{SN}$. Therefore all $m, s, E[x] \in \mathsf{SN}$ which by properties of SN implies $\mathsf{It}(m,s,E[x]) \in \mathsf{SN}$.
- Assume $s \in \Phi(\mathcal{M} \times \mathcal{N}) \to \mathcal{N}$. The goal is $\mathsf{Rec}(m,s,E[x]) \in \mathcal{N}$. As in the previous case we obtain $m, s \in \mathsf{SN}$, therefore by properties of SN we conclude $E'[x] := \mathsf{Rec}(m,s,E[x]) \in \mathsf{SN}$. Therefore, as $\mathcal{N} \in \mathsf{SAT}$ we get $E'[x] \in \mathcal{N}$.

The other closure rules for SAT sets are proved in a similar way. □

Corollary 3. *$\mathcal{E}_{\mu}(\mathcal{M}) = \Psi_{E}(\mathcal{M})$.*

Proof. $\subseteq$). we have $\mathcal{E}_{\mu}(\mathcal{M}) = \mathcal{E}_{\mu}(\mathcal{M}) \cap \mathsf{SN} \subseteq \mathsf{cl}(\mathcal{E}_{\mu}(\mathcal{M})) \equiv \Psi_{E}(\mathcal{M})$.
$\supseteq$). By the previous lemma we have $\mathcal{E}_{\mu}(\mathcal{M}) \in \mathsf{SAT}$. Therefore by minimality of the closure we get $\Psi_{E}(\mathcal{M}) \equiv \mathsf{cl}(\mathcal{E}_{\mu}(\mathcal{M})) \subseteq \mathcal{E}_{\mu}(\mathcal{M})$.

□

Lemma 4. *$\Psi_{I}(\mathcal{M}) \subseteq \mathcal{M} \Leftrightarrow \forall t \in \Phi(\mathcal{M}).\ \mathsf{in}\, t \in \mathcal{M}$.*

Proof. $\Rightarrow$) Assume $\Psi_{I}(\mathcal{M}) \subseteq \mathcal{M}$, i.e., $\mathsf{cl}(\mathcal{I}_{\mu}(\mathcal{M})) \subseteq \mathcal{M}$. Take $t \in \Phi(\mathcal{M})$, this implies $\mathsf{in}\, t \in \mathcal{I}_{\mu}(\mathcal{M})$, which, by corollary 2, implies $\mathsf{in}\, t \in \Psi_{I}(\mathcal{M}) \subseteq \mathcal{M}$. Therefore $\mathsf{in}\, t \in \mathcal{M}$.
$\Leftarrow$) Assume $\forall t \in \Phi(\mathcal{M}).\ \mathsf{in}\, t \in \mathcal{M}$ and take $r \in \Psi_{I}(\mathcal{M}) \equiv \mathsf{cl}(\mathcal{I}_{\mu}(\mathcal{M}))$. Goal is $r \in \mathcal{M}$. As $\mathcal{M} \in \mathsf{SAT}$ it suffices to show $\mathcal{I}_{\mu}(\mathcal{M}) \cap \mathsf{SN} \subseteq \mathcal{M}$, the goal follows by minimality of the closure. By lemma 2 we have $\mathcal{I}_{\mu}(\mathcal{M}) \subseteq \mathsf{SN}$, thus we only have to show $\mathcal{I}_{\mu}(\mathcal{M}) \subseteq \mathcal{M}$. Take $\mathsf{in}\, t \in \mathcal{I}_{\mu}(\mathcal{M})$, so $t \in \Phi(\mathcal{M})$ which by assumption implies $\mathsf{in}\, t \in \mathcal{M}$. Therefore $\mathcal{I}_{\mu}(\mathcal{M}) \subseteq \mathcal{M}$. □

Lemma 5.

$$\begin{array}{rl}\mathcal{M} \subseteq \Psi_E(\mathcal{M}) \Leftrightarrow & \forall r \in \mathcal{M}.\forall m \in \mathsf{mon}(\Phi).\,\forall \mathcal{N} \in \mathsf{SAT}.\\ & \big(\forall s \in \Phi(\mathcal{N}) \to \mathcal{N}.\ \mathsf{It}(m,s,r) \in \mathcal{N}\big) \wedge \\ & \big(\forall s \in \Phi(\mathcal{M}\times\mathcal{N}) \to \mathcal{N}.\ \mathsf{Rec}(m,s,r) \in \mathcal{N}\big)\end{array}$$

Proof. Call $\Box(r)$ to the condition on the right hand side for a given $r \in \mathcal{M}$.
$\Rightarrow$). Assume $\mathcal{M} \subseteq \Psi_E(\mathcal{M})$. We have to show $\Box(r)$ for all $r \in \mathcal{M}$. Take $r \in \mathcal{M}$, by corollary 3 we have $\mathcal{M} \subseteq \mathcal{E}_\mu(\mathcal{M})$. Observing that $\mathcal{E}_\mu(\mathcal{M}) = \{r \in \mathsf{SN} |\ \Box(r)\}$ we are done.
$\Leftarrow$) Assume $\forall r \in \mathcal{M}.\Box(r)$ and take $r \in \mathcal{M}$, we have to show that $r \in \Psi_E(M)$. By corollary 3 suffices to show that $r \in \mathcal{E}_\mu(\mathcal{M})$. We have $r \in \mathsf{SN}$ because $\mathcal{M} \subseteq \mathsf{SN}$. Moreover $\Box(r)$ holds by assumption, which implies $r \in \mathcal{E}_\mu(\mathcal{M})$. □

Lemma 6. *$\mu(\Phi)$ is a pre-fixed point of Ψ_I. i.e., $\Psi_I\big(\mu(\Phi)\big) \subseteq \mu(\Phi)$*

Proof. By definition of $\mu(\Phi)$ it suffices to show $\Psi_I\Big(\Psi_I^{\supseteq}\big(\mu(\Phi)\big)\Big) \subseteq \Psi_I^{\supseteq}\big(\mu(\Phi)\big)$, to show this we will use the lemma 4. Take $t \in \Phi\Big(\Psi_I^{\supseteq}\big(\mu(\Phi)\big)\Big)$, this implies $\mathsf{in}\, t \in \mathcal{I}_\mu^{\supseteq}\Big(\Psi_I^{\supseteq}\big(\mu(\Phi)\big)\Big) \subseteq \Psi_I^{\supseteq}\Big(\Psi_I^{\supseteq}\big(\mu(\Phi)\big)\Big)$, the last inclusion given by corollary 2. Therefore by definition of $\mu(\Phi)$ we conclude $\mathsf{in}\, t \in \Psi_I^{\supseteq}\big(\mu(\Phi)\big)$. □

Lemma 7. *$\mu(\Phi)$ is a post-fixed point of Ψ_E. i.e., $\mu(\Phi) \subseteq \Psi_E(\mu(\Phi))$*

Proof. Our goal is $\mu(\Phi) \subseteq \Psi_E\big(\mu(\Phi)\big)$. To prove this we will use extended induction on $\mu(\Phi)$. Therefore the goal becomes $\Psi_I^{\supseteq}\Big(\mu(\Phi) \cap \Psi_E\big(\mu(\Phi)\big)\Big) \subseteq \Psi_E\big(\mu(\Phi)\big)$.

Set $\mathcal{L} := \mu(\Phi)$, $\mathcal{L}' := \mathcal{L} \cap \Psi_E(\mathcal{L})$. The goal is $\Psi_I^{\supseteq}(\mathcal{L}') \subseteq \Psi_E(\mathcal{L})$. By monotonicity of the closure it suffices to show $\mathcal{I}_\mu^{\supseteq}(\mathcal{L}') \subseteq \mathcal{E}_\mu(\mathcal{L})$. Take $t \in \mathcal{I}_\mu^{\supseteq}(\mathcal{L}')$, i.e., $t \equiv \mathsf{in}\, r$ with $r \in \Phi^{\supseteq}(\mathcal{L}')$. We need to show $\mathsf{in}\, r \in \mathcal{E}_\mu(\mathcal{L})$. First observe that $\mathsf{in}\, r \in \mathsf{SN}$ because $r \in \Phi^{\supseteq}(\mathcal{L}') \subseteq \mathsf{SN}$ and by properties of SN. Next we have to prove that $\Box(\mathsf{in}\, r)$ (cf. proof of lemma 5), so fix $m \in \mathsf{mon}(\Phi)$ and $\mathcal{N} \in \mathsf{SAT}$.

- Take $s \in \Phi(\mathcal{N}) \to \mathcal{N}$. We want to show that $\mathsf{It}(m,s,\mathsf{in}\, r) \in \mathcal{N}$. Using that $\mathcal{N} \in \mathsf{SAT}$, it suffices to show that $s\big(m\big(\lambda x.\mathsf{It}(m,s,x)\big)r\big) \in \mathcal{N}$. As $s \in \Phi(\mathcal{N}) \to \mathcal{N}$ we only have to show $m\big(\lambda x.\mathsf{It}(m,s,x)\big)r \in \Phi(\mathcal{N})$ but observing that $r \in \Phi^{\supseteq}(\mathcal{L}')$ we only have to show that $m \in \mathsf{mon}(\Phi), \mathcal{N} \in \mathsf{SAT}$ and $\lambda x.\mathsf{It}(m,s,x) \in \mathcal{L}' \to \mathcal{N}$. The first two claims are given and to prove the last one we will show that $\mathsf{It}(m,s,x) \in \mathsf{S}_x(\mathcal{L}',\mathcal{N})$. Take $q \in \mathcal{L}'$ we prove $\mathsf{It}(m,s,x)[x := q] \in \mathcal{N}$, w.l.o.g. $x \notin FV(m,s)$ therefore we show $\mathsf{It}(m,s,q) \in \mathcal{N}$. We have $\mathcal{L}' \subseteq \Psi_E(\mathcal{L}) = \mathcal{E}_\mu(\mathcal{L})$, the equality given by corollary 3. Therefore $q \in \mathcal{E}_\mu(\mathcal{L})$ which immediately yields $\mathsf{It}(m,s,q) \in \mathcal{N}$.
- Take $s \in \Phi(\mathcal{L}\times\mathcal{N}) \to \mathcal{N}$. We need to prove $\mathsf{Rec}(m,s,\mathsf{in}\, r) \in \mathcal{N}$. By a similar reason as the previous case we only have to show

$$\lambda z.\langle(\lambda yy)z, (\lambda x.\mathsf{Rec}(m,s,x))z\rangle \in \mathcal{L}' \to \mathcal{L}\times\mathcal{N}.$$

  It suffices to prove $\langle(\lambda yy)z, (\lambda x.\mathsf{Rec}(m,s,x))z\rangle \in \mathsf{S}_z(\mathcal{L}', \mathcal{L}\times\mathcal{N})$, so we take $q \in \mathcal{L}'$ and show $\langle(\lambda yy)q, (\lambda x.\mathsf{Rec}(m,s,x))q\rangle \in \mathcal{L}\times\mathcal{N}$. For this we prove two things:
  - $(\lambda yy)q \in \mathcal{L}$. Clearly we have $\lambda yy \in \mathcal{L} \to \mathcal{L}$ and as $q \in \mathcal{L}' \subseteq \mathcal{L}$ we get $(\lambda yy)q \in \mathcal{L}$.
  - $(\lambda x.\mathsf{Rec}(m,s,x))q \in \mathcal{N}$. It suffices to show $\lambda x.\mathsf{Rec}(m,s,x) \in \mathcal{L}' \to \mathcal{N}$, that is $\mathsf{Rec}(m,s,x) \in \mathsf{S}_x(\mathcal{L}',\mathcal{N})$. Take $p \in \mathcal{L}'$, we will show $\mathsf{Rec}(m,s,x)[x := p] \in \mathcal{N}$, where w.l.o.g. $x \notin FV(m,s)$ so we prove $\mathsf{Rec}(m,s,p) \in \mathcal{N}$. We have $\mathcal{L}' \subseteq \Psi_E(\mathcal{L}) = \mathcal{E}_\mu(\mathcal{L})$, the equality given by corollary 3. Therefore $p \in \mathcal{E}_\mu(\mathcal{L})$ which immediately yields $\mathsf{Rec}(m,s,p) \in \mathcal{N}$.

Therefore $\Box(\mathsf{in}\, r)$ and we are done. □

3.4.4. *Saturated Sets for Coinductive Types*

These constructions for saturated sets are, to our knowledge, new.

Definition 16. *Given $\Phi : \mathsf{SAT} \to \mathsf{SAT}, \mathcal{M} \in \mathsf{SAT}$, define*

$$\begin{array}{rcl}\mathcal{I}_\nu(\mathcal{M}) & := & \{\mathsf{CoIt}(m,s,t) \mid m \in \mathsf{mon}(\Phi),\ s \in \mathcal{N} \to \Phi(\mathcal{N}),\ t \in \mathcal{N}, \mathcal{N} \in \mathsf{SAT}\}\\ & \cup & \{\mathsf{CoRec}(m,s,t) \mid m \in \mathsf{mon}(\Phi),\ s \in \mathcal{N} \to \Phi(\mathcal{M}+\mathcal{N}),\ t \in \mathcal{N} \in \mathsf{SAT}\}\\ & \cup & \{\mathsf{out}^{-1}(m,t) \mid m \in \mathsf{mon}(\Phi),\ t \in \Phi(\mathcal{M})\}\end{array}$$

and $\Psi_I : \mathsf{SAT} \to \mathsf{SAT}$ with $\Psi_I(\mathcal{M}) := \mathsf{cl}(\mathcal{I}_\nu(\mathcal{M}))$.

Lemma 8. *$\mathcal{I}_\nu(\mathcal{M}) \subseteq \mathsf{SN}$.*

Proof. Take $r \in \mathcal{I}_\nu(\mathcal{M})$. We have three cases:

- $r \equiv \mathsf{CoIt}(m,s,t)$. We have $m,s,t \in \mathsf{SN}$ because they belong to some saturated set. Therefore by properties of SN we also have $\mathsf{CoIt}(m,s,t) \in \mathsf{SN}$.
- $r \equiv \mathsf{CoRec}(m,s,t)$. Similarly $m,s,t \in \mathsf{SN}$ implies $\mathsf{CoRec}(m,s,t) \in \mathsf{SN}$.
- $r \equiv \mathsf{out}^{-1}(m,t)$. Again $m,t \in \mathsf{SN}$ implies $\mathsf{out}^{-1}(m,t) \in \mathsf{SN}$.

□

Corollary 4. $\mathcal{I}_\nu(\mathcal{M}) \subseteq \Psi_I(\mathcal{M})$.

Proof. By definition of closure we have $\mathcal{I}_\nu(\mathcal{M}) \cap \mathsf{SN} \subseteq \mathsf{cl}(\mathcal{I}_\nu(\mathcal{M}))$ which, by the previous lemma is the same as $\mathcal{I}_\nu(\mathcal{M}) \subseteq \mathsf{cl}(\mathcal{I}_\nu(\mathcal{M})) \equiv \Psi_I(\mathcal{M})$. □

Definition 17. *Given* $\Phi : \mathsf{SAT} \to \mathsf{SAT}, \mathcal{M} \in \mathsf{SAT}$, *define* $\mathcal{E}_\nu(\mathcal{M}) := \{r \in \mathsf{SN} \mid \mathsf{out}\, r \in \Phi(\mathcal{M})\}$ *and* $\Psi_E : \mathsf{SAT} \to \mathsf{SAT}$, *with* $\Psi_E(\mathcal{M}) := \mathsf{cl}(\mathcal{E}_\nu(\mathcal{M}))$

Again we do not know if Ψ_E is monotone and proceed as follows: define $\Phi^{\subseteq} : \mathsf{SAT} \to \mathsf{SAT}$ as $\Phi^{\subseteq}(\mathcal{M}) := \mathsf{cl}(\mathsf{A}(\mathcal{M}))$ with

$$\mathsf{A}(\mathcal{M}) := \{mqr \mid m \in \mathsf{mon}(\Phi),\ q \in \mathcal{N} \to \mathcal{M},\ r \in \Phi(\mathcal{N}) \text{ for some } \mathcal{N} \in \mathsf{SAT}\}$$

Lemma 9. *For all* $\mathcal{M} \in \mathsf{SAT}$, $\mathsf{A}(\mathcal{M}) \subseteq \Phi(\mathcal{M})$.

Proof. Take $t \in \mathsf{A}(\mathcal{M})$, i.e.,$t \equiv mqr$ with $m \in \mathsf{mon}(\Phi),\ q \in \mathcal{N} \to \mathcal{M},\ r \in \Phi(\mathcal{N})$ for some $\mathcal{N} \in \mathsf{SAT}$. $m \in \mathsf{mon}(\Phi) \Rightarrow m \in (\mathcal{N} \to \mathcal{M}) \to (\Phi(\mathcal{N}) \to \Phi(\mathcal{M})) \Rightarrow mq \in \Phi(\mathcal{N}) \to \Phi(\mathcal{M}) \Rightarrow mqr \in \Phi(\mathcal{M})$, i.e. $t \in \Phi(\mathcal{M})$. □

Corollary 5. *For all* $\mathcal{M} \in \mathsf{SAT}$, $\mathsf{A}(\mathcal{M}) \subseteq \mathsf{SN}$.

Proof. $\mathsf{A}(\mathcal{M}) \subseteq \Phi(\mathcal{M}) \subseteq \mathsf{SN}$. □

Corollary 6. *For all* $\mathcal{M} \in \mathsf{SAT}$, $\Phi^{\subseteq}(\mathcal{M}) \subseteq \Phi(\mathcal{M})$.

Proof. As $\Phi(\mathcal{M}) \in \mathsf{SAT}$, by minimality of the closure it suffices to show $\mathsf{A}(\mathcal{M}) \cap \mathsf{SN} \subseteq \Phi(\mathcal{M})$, but by the previous corollary we only need to show $\mathsf{A}(\mathcal{M}) \subseteq \Phi(\mathcal{M})$ but this is the statement of the lemma. □

Corollary 7. *For all* $\mathcal{M} \in \mathsf{SAT}$, $\mathsf{A}(\mathcal{M}) \subseteq \Phi^{\subseteq}(\mathcal{M})$.

Proof. $\mathsf{A}(\mathcal{M}) = \mathsf{A}(\mathcal{M}) \cap \mathsf{SN} \subseteq \mathsf{cl}(\mathsf{A}(\mathcal{M})) \equiv \Phi^{\subseteq}(\mathcal{M})$. □

Lemma 10. *For all* $\mathcal{P}, \mathcal{Q}, \mathcal{N} \in \mathsf{SAT}$. *If* $\mathcal{P} \subseteq \mathcal{Q}$ *then* $\mathcal{N} \to \mathcal{P} \subseteq \mathcal{N} \to \mathcal{Q}$.

Proof. It suffices to show that $\mathcal{I}_\to(\mathcal{N},\mathcal{P}) \cap \mathsf{SN} = \mathcal{I}_\to(\mathcal{N},\mathcal{P}) \subseteq \mathcal{I}_\to(\mathcal{N},\mathcal{Q})$. Take $\lambda xt \in \mathcal{I}_\to(\mathcal{N},\mathcal{P})$, i.e., $t \in \mathsf{S}_x(\mathcal{N},\mathcal{P})$. Therefore we have $\forall s \in \mathcal{N}.t[x := s] \in \mathcal{P}$ which by assumption implies $\forall s \in \mathcal{N}.t[x := s] \in \mathcal{Q}$. That is $t \in \mathsf{S}_x(\mathcal{N},\mathcal{Q}) \Rightarrow \lambda xt \in \mathcal{I}_\to(\mathcal{N},\mathcal{Q})$. □

Corollary 8. $\Phi^{\subseteq}$ *is monotone, i.e., for all* $\mathcal{P}, \mathcal{Q} \in \mathsf{SAT}$, *if* $\mathcal{P} \subseteq \mathcal{Q}$ *then* $\Phi^{\subseteq}(\mathcal{P}) \subseteq \Phi^{\subseteq}(\mathcal{Q})$.

Proof. Assume $\mathcal{P} \subseteq \mathcal{Q}$. Take $mqr \in \Phi^{\subseteq}(\mathcal{P})$, then $m \in \mathsf{mon}(\Phi),\ q \in \mathcal{N} \to \mathcal{P},\ r \in \Phi(\mathcal{N})$. $q \in \mathcal{N} \to \mathcal{P}$ implies by the previous lemma $q \in \mathcal{N} \to \mathcal{Q}$. Therefore we have $mqr \in \Phi^{\subseteq}(\mathcal{Q})$. □

Next set $\mathcal{E}^{\subseteq}_\nu(\mathcal{M}) := \{r \in \mathsf{SN} \mid \mathsf{out}\, r \in \Phi^{\subseteq}(\mathcal{M})\}$ and define $\Psi^{\subseteq}_E : \mathsf{SAT} \to \mathsf{SAT}$ as $\Psi^{\subseteq}_E(\mathcal{M}) := \mathsf{cl}(\mathcal{E}^{\subseteq}_\nu(\mathcal{M}))$.

Clearly $\Psi^{\subseteq}_E$ is monotone, because so is $\Phi^{\subseteq}$, therefore the following definition is valid:

$$\nu(\Phi) := \mathsf{gfp}(\Psi^{\subseteq}_E).$$

i.e., $\nu(\Phi)$ is the greatest fixed point of $\Psi^{\subseteq}_E$.

Lemma 11. $\mathcal{E}_\nu(\mathcal{M}),\ \mathcal{E}^{\subseteq}_\nu(\mathcal{M}) \in \mathsf{SAT}$.

Proof. We show the first part. Clearly we have $\mathcal{E}_\nu(\mathcal{M}) \subseteq \mathsf{SN}$.
Take $E[x] \in \mathsf{SN}$. Goal is $E[x] \in \mathcal{E}_\nu(\mathcal{M})$, i.e., $\mathsf{out}\,\mathcal{E}[x] \in \Phi(\mathcal{M})$. By properties of SN, $E[x] \in \mathsf{SN}$ implies $\mathsf{out}\,\mathcal{E}[x] \in \mathsf{SN}$, but $\mathsf{out}\,\mathcal{E}[x]$ is a multiple elimination say $E'[x] \in \mathsf{SN}$. Therefore, as $\Phi(\mathcal{M}) \in \mathsf{SAT}$, we get $E'[x] \in \Phi(\mathcal{M})$. The remaining rules are easily proved.

□

Corollary 9. $\mathcal{E}_\nu(\mathcal{M}) = \Psi_E(\mathcal{M}),\ \mathcal{E}^{\subseteq}_\nu(\mathcal{M}) = \Psi^{\subseteq}_E(\mathcal{M})$

Proof. We show the first part.
$\subseteq$) We have $\mathcal{E}_\nu(\mathcal{M}) \cap \mathsf{SN} \subseteq \mathsf{cl}(\mathcal{E}_\nu(\mathcal{M}))$, which, as $\mathcal{E}_\nu(\mathcal{M}) \subseteq \mathsf{SN}$, is the same as $\mathcal{E}_\nu(\mathcal{M}) \subseteq \mathsf{cl}(\mathcal{E}_\nu(\mathcal{M})) \equiv \Psi_E(\mathcal{M})$.
$\supseteq$). By the previous lemma, using the minimality of the closure we have $\Psi_E(\mathcal{M}) = \mathsf{cl}(\mathcal{E}_\nu(\mathcal{M})) \subseteq \mathcal{E}_\nu(\mathcal{M})$. □

Lemma 12. $\mathcal{M} \subseteq \Psi_E(\mathcal{M}) \Leftrightarrow \forall t \in \mathcal{M}.\, \mathsf{out}\, t \in \Phi(\mathcal{M})$

Proof. $\Rightarrow$) Take $t \in \mathcal{M}$, by assumption we get $t \in \Psi_E(\mathcal{M})$, and by the previous corollary $t \in \mathcal{E}_\nu(\mathcal{M})$, which by definition of $\mathcal{E}_\nu(\mathcal{M})$ yields $\mathsf{out}\, t \in \Phi(\mathcal{M})$.
$\Leftarrow$) Take $t \in \mathcal{M}$, by assumption we get $\mathsf{out}\, t \in \Phi(\mathcal{M})$. On the other hand, as $\mathcal{M} \subseteq \mathsf{SN}$, we get $t \in \mathsf{SN}$. Therefore $t \in \mathcal{E}_\nu(\mathcal{M})$, which by the previous corollary is the same as $t \in \Psi_E(\mathcal{M})$. □

Lemma 13.

$$\begin{aligned}\Psi_I(\mathcal{M}) \subseteq \mathcal{M} \Leftrightarrow \quad & \forall m \in \mathsf{mon}(\Phi).\, \forall \mathcal{N} \in \mathsf{SAT}. \\ & \big(\forall t \in \mathcal{N}\ \forall s \in \mathcal{N} \to \Phi(\mathcal{N}).\, \mathsf{CoIt}(m,s,t) \in \mathcal{M}\big) \wedge \\ & \big(\forall t \in \mathcal{N}\ \forall s \in \mathcal{N} \to \Phi(\mathcal{M}+\mathcal{N}).\, \mathsf{CoRec}(m,s,t) \in \mathcal{M}\big) \wedge \\ & \big(\forall t \in \Phi(\mathcal{M}).\, \mathsf{out}^{-1}(m,t) \in \mathcal{M}\big)\end{aligned}$$

Proof. $\Rightarrow$). Assume $\Psi_I(\mathcal{M}) \subseteq \mathcal{M}$. By corollary 4 we get $\mathcal{I}_\nu(\mathcal{M}) \subseteq \mathcal{M}$.
Take $m \in \mathsf{mon}(\Phi), \mathcal{N} \in \mathsf{SAT}$. We prove every part of the conjunction:

- Take $t \in \mathcal{N},\ s \in \mathcal{N} \to \Phi(\mathcal{N})$. From this we get $\mathsf{CoIt}(m,s,t) \in \mathcal{I}_\nu(\mathcal{M})$, therefore $\mathsf{CoIt}(m,s,t) \in \mathcal{M}$.
- Take $t \in \mathcal{N}, s \in \mathcal{N} \to \Phi(\mathcal{M}+\mathcal{N})$. Analogously to the previous case we get $\mathsf{CoRec}(m,s,t) \in \mathcal{I}_\nu(\mathcal{M}) \subseteq \mathcal{M}$.
- Take $t \in \Phi(\mathcal{M})$. This yields $\mathsf{out}^{-1}(m,t) \in \mathcal{I}_\nu(\mathcal{M}) \subseteq \mathcal{M}$.

$\Leftarrow$) Assume the condition on the right hand side. We have $\Psi_I(M) = \mathsf{cl}(\mathcal{I}_\nu(\mathcal{M}))$. By minimality of the closure it suffices to show $\mathcal{I}_\nu(\mathcal{M}) \cap \mathsf{SN} \subseteq \mathcal{M}$. But by lemma 8 this is the same as $\mathcal{I}_\nu(\mathcal{M}) \subseteq \mathcal{M}$. But this follows immediately from the assumption and the definition of $\mathcal{I}_\nu(\mathcal{M})$. □

Lemma 14. *$\nu(\Phi)$ is a pre-fixed point of Ψ_I. i.e., $\Psi_I(\nu(\Phi)) \subseteq \nu(\Phi)$*

Proof. We will use extended coinduction. Therefore the goal becomes

$$\Psi_I\big(\nu(\Phi)\big) \subseteq \Psi_E^{\subseteq}\Big(\nu(\Phi) \cup \Psi_I\big(\nu(\Phi)\big)\Big)$$

Set $\mathcal{G} := \nu(\Phi)$, $\mathcal{G}' := \mathcal{G} \cup \Psi_I(\mathcal{G})$. The goal becomes $\Psi_I(\mathcal{G}) \subseteq \Psi_E^{\subseteq}(\mathcal{G}')$. By monotonicity of the closure it suffices to show $\mathcal{I}_\nu(\mathcal{G}) \subseteq \mathcal{E}_\nu^{\subseteq}(\mathcal{G}')$. Assume $r \in \mathcal{I}_\nu(\mathcal{G})$. To show $r \in \mathcal{E}_\nu^{\subseteq}(\mathcal{G}')$ it suffices $\mathsf{out}\, r \in \Phi^{\subseteq}(\mathcal{G}')$ ($r \in \mathsf{SN}$ because $\mathcal{I}_\nu(\mathcal{G}) \subseteq \mathsf{SN}$). We have three cases:

- $r \equiv \mathsf{CoIt}(m,s,t)$ with $m \in \mathsf{mon}(\Phi), s \in \mathcal{N} \to \Phi(\mathcal{N}), t \in \mathcal{N}$. By properties of saturated sets it suffices to show $m\big(\lambda z.\mathsf{CoIt}(m,s,z)\big)(st) \in \Phi^{\subseteq}(\mathcal{G}')$ and using corollary 7 we will prove only $m\big(\lambda z.\mathsf{CoIt}(m,s,z)\big)(st) \in \mathsf{A}(\mathcal{G}')$. We have by assumption $m \in \mathsf{mon}(\Phi)$ and easily we get $st \in \Phi(\mathcal{N})$. To prove $\lambda z.\mathsf{CoIt}(m,s,z) \in \mathcal{N} \to \mathcal{G}'$, we show $\mathsf{CoIt}(m,s,z) \in \mathsf{S}_z(\mathcal{N},\mathcal{G}')$. Taking $q \in \mathcal{N}$ we show $\mathsf{CoIt}(m,s,z)[z := q] \equiv \mathsf{CoIt}(m,s,q) \in \mathcal{G}'$. Clearly $\mathsf{CoIt}(m,s,q) \in \mathcal{I}_\nu(\mathcal{G})$, therefore by corollary 4 we have $\mathsf{CoIt}(m,s,q) \in \Psi_I(\mathcal{G}) \subseteq \mathcal{G}'$.
- $r \equiv \mathsf{CoRec}(m,s,t)$ with $m \in \mathsf{mon}(\Phi), s \in \mathcal{N} \to \Phi(\mathcal{G}+\mathcal{N}), t \in \mathcal{N}$. By similar reasoning as the previous case we only need to show

  $$m\big([\mathsf{Id}, \lambda z.\mathsf{CoRec}(m,s,z)]\big)(st) \in \mathsf{A}(\mathcal{G}').$$

  We have $m \in \mathsf{mon}(\Phi)$ and easily we get $st \in \Phi(\mathcal{G}+\mathcal{N})$. Remains to show that $[\mathsf{Id}, \lambda z.\mathsf{CoRec}(m,s,z)] \in \mathcal{G}+\mathcal{N} \to \mathcal{G}'$. We have $[\mathsf{Id}, \lambda z.\mathsf{CoRec}(m,s,z)] \equiv \lambda x.\mathsf{case}(x, y.y, z.\mathsf{CoRec}(m,s,z))$ therefore the goal reduces to show $\mathsf{case}(x, y.y, z.\mathsf{CoRec}(m,s,z)) \in \mathsf{S}_x(\mathcal{G}+\mathcal{N},\mathcal{G}')$. So we take $q \in \mathcal{G}+\mathcal{N}$ and prove $\mathsf{case}(x, y.y, z.\mathsf{CoRec}(m,s,z)) \in \mathcal{G}'$, which, by properties of saturated sets, reduces to the next two claims:
  - $y \in \mathsf{S}_y(\mathcal{G},\mathcal{G}')$. This holds trivially because $\mathcal{G} \subseteq \mathcal{G}'$.
  - $\mathsf{CoRec}(m,s,z) \in \mathsf{S}_z(\mathcal{N},\mathcal{G}')$. For this we take $p \in \mathcal{N}$ and show $\mathsf{CoRec}(m,s,z)[z := p] \equiv \mathsf{CoRec}(m,s,p) \in \mathcal{G}'$. Clearly we have $\mathsf{CoRec}(m,s,p) \in \mathcal{I}_\nu(\mathcal{G})$. Therefore by corollary 4 we have $\mathsf{CoRec}(m,s,p) \in \Psi_I(\mathcal{G}) \subseteq \mathcal{G}'$.

- $r \equiv \mathsf{out}^{-1}(m,t)$ with $m \in \mathsf{mon}(\Phi)$ and $t \in \Phi(\mathcal{G})$. By properties of saturated sets it suffices to show $m(\lambda zz)t \in \Phi^{\subseteq}(\mathcal{G}')$. Using corollary 7 we show $m(\lambda zz)t \in \mathsf{A}(\mathcal{G}')$. We have $m \in \mathsf{mon}(\Phi)$ and $t \in \Phi(\mathcal{G})$, only remains to show $\lambda zz \in \mathcal{G} \to \mathcal{G}'$, but this is consequence of $\mathcal{G} \subseteq \mathcal{G}'$.

□

Lemma 15. *$\nu(\Phi)$ is a post-fixed point of Ψ_E. i.e., $\nu(\Phi) \subseteq \Psi_E(\nu(\Phi))$*

Proof. By lemma 12 it suffices to show $\forall t \in \nu(\Phi).\, \mathsf{out}\, t \in \Phi\big(\nu(\Phi)\big)$. By definition we have $\nu(\Phi) = \Psi_E^{\subseteq}\big(\nu(\Phi)\big)$ and by corollary 9 $\Psi_E^{\subseteq}\big(\nu(\Phi)\big) = \mathcal{E}_\nu^{\subseteq}\big(\nu(\Phi)\big)$. So take $t \in \nu(\Phi) = \mathcal{E}_\nu^{\subseteq}\big(\nu(\Phi)\big) \Rightarrow \mathsf{out}\, t \in \Phi^{\subseteq}\big(\nu(\Phi)\big)$. Finally by corollary 6 we get $\mathsf{out}\, t \in \Phi\big(\nu(\Phi)\big)$. □

Proposition 6 (Soundness of the constructions)**.** *Given $\Phi : \mathsf{SAT} \to \mathsf{SAT}$ the following holds.*

1. *If $t \in \Phi(\mu(\Phi))$ then $\mathsf{in}\, t \in \mu(\Phi)$.*
2. *If $r \in \mu(\Phi), m \in \mathsf{mon}(\Phi), \mathcal{N} \in \mathsf{SAT}$ and $s \in \Phi(\mathcal{N}) \to \mathcal{N}$ then $\mathsf{It}(m,s,r) \in \mathcal{N}$.*
3. *If $r \in \mu(\Phi), m \in \mathsf{mon}(\Phi), \mathcal{N} \in \mathsf{SAT}$ and $s \in \Phi(\mu(\Phi) \times \mathcal{N}) \to \mathcal{N}$ then $\mathsf{Rec}(m,s,r) \in \mathcal{N}$.*
4. *If $t \in \nu(\Phi)$ then $\mathsf{out}\, t \in \Phi(\nu(\Phi))$.*
5. *If $\mathcal{N} \in \mathsf{SAT}, r \in \mathcal{N}, m \in \mathsf{mon}(\Phi)$ and $s \in \mathcal{N} \to \Phi(\mathcal{N})$ then $\mathsf{CoIt}(m,s,r) \in \nu(\Phi)$.*
6. *If $\mathcal{N} \in \mathsf{SAT}, r \in \mathcal{N}, m \in \mathsf{mon}(\Phi)$ and $s \in \mathcal{N} \to \Phi(\nu(\Phi) + \mathcal{N})$ then $\mathsf{CoRec}(m,s,r) \in \nu(\Phi)$.*
7. *If $m \in \mathsf{mon}(\Phi)$ and $r \in \Phi(\nu(\Phi))$ then $\mathsf{out}^{-1}(m,r) \in \nu(\Phi)$.*

Proof. For part 1, use lemmas 6 and 4. For parts 2, 3 use lemmas 7 and 5. Part 4 follows from lemmas 15 and 12. Finally to prove parts 5, 6, 7 use lemmas 14 and 13.

□

The remainder of the proof of strong normalization follows a standard technique: first we use saturated sets to define the computability predicates which give a semantics to the types. Then we prove substitution and coincidence lemmas for the predicates and show that every typable term lies in the predicate of its type and hence belongs to SN, which proves strong normalization.

Definition 18. *A candidate assignment is a finite set of pairs of the form $X : \mathcal{M}$ where X is a type variable and $\mathcal{M} \in \mathsf{SAT}$ such that no type variable occurs twice. Candidate assignments are denoted with Γ, in the expression $\Gamma, X : \mathcal{M}$ is understood that $X \notin \Gamma$.*

Definition 19 (Strong Computability Predicates)**.** *Given a type A and a candidate assigment Γ we define the saturated set of strongly computable terms with respect to A and Γ,denoted $\mathsf{SC}^A[\Gamma]$, as follows:*

$$
\begin{aligned}
\mathsf{SC}^X[\Gamma] &:= \begin{cases} \mathcal{M} & \text{if } X : \mathcal{M} \in \Gamma \\ \mathsf{SN} & \text{otherwise.} \end{cases} \\
\mathsf{SC}^{A\to B}[\Gamma] &:= \mathsf{SC}^A[\Gamma] \to \mathsf{SC}^B[\Gamma] \\
\mathsf{SC}^{A+B}[\Gamma] &:= \mathsf{SC}^A[\Gamma] + \mathsf{SC}^B[\Gamma] \\
\mathsf{SC}^{A\times B}[\Gamma] &:= \mathsf{SC}^A[\Gamma] \times \mathsf{SC}^B[\Gamma] \\
\mathsf{SC}^{\forall XA}[\Gamma] &:= \textstyle\bigcap_{\mathcal{M}\in\mathsf{SAT}} \mathsf{SC}^A[\Gamma, X : \mathcal{M}] \\
\mathsf{SC}^{\mu XA}[\Gamma] &:= \mu(\Phi_\Gamma^{\lambda XA}) \\
\mathsf{SC}^{\nu XA}[\Gamma] &:= \nu(\Phi_\Gamma^{\lambda XA})
\end{aligned}
$$

where $\Phi_\Gamma^{\lambda XA} : \mathsf{SAT} \to \mathsf{SAT}$ is defined as:

$$\Phi_\Gamma^{\lambda XA}(\mathcal{M}) := \mathsf{SC}^A[\Gamma, X : \mathcal{M}]$$

Lemma 16 (Coincidence and Substitution)**.** *The following properties hold:*

- *If $X \notin FV(A)$ then $\mathsf{SC}^A[\Gamma, X : \mathcal{M}] = \mathsf{SC}^A[\Gamma]$.*
- *$\mathsf{SC}^{A[X:=B]}[\Gamma] = \mathsf{SC}^A[\Gamma, X : \mathsf{SC}^B[\Gamma]]$.*

Proof. Induction on A.

□

Lemma 17 (Main Lemma)**.** *If* $\Gamma \vdash r : A$ *with* $\Gamma = \{x_1 : A_1, \ldots, x_k : A_k\}$ *and for all for* $1 \leq i \leq k$ *we have* $s_i \in \mathsf{SC}^{A_i}[\Gamma]$ *then* $r[\vec{x} := \vec{s}\,] \in \mathsf{SC}^{A}[\Gamma]$.

Proof. Induction on $\vdash$.

□

Proposition 7. *If* $\Gamma \vdash r : A$ *then* $r \in \mathsf{SN}$.

Proof. Assume $\Gamma = \{x_1 : A_1, \ldots, x_k : A_k\}$. As the set of variables is contained in every saturated set we have $x_i \in \mathsf{SC}^{A_i}[\varnothing]$ therefore as $\Gamma \vdash r : A$ the main lemma yields $r[\vec{x} := \vec{x}] \in \mathsf{SC}^{A}[\varnothing] \subseteq \mathsf{SN}$. Therefore $r \in \mathsf{SN}$. □

Proposition 8. MICT *is strongly normalizing.*

Proof. If $\Gamma \vdash r : A$ then proposition 7 yields $r \in \mathsf{SN}$. But by proposition 3 we have $\mathsf{SN} \subseteq \mathsf{sn}$. Therefore $r \in \mathsf{sn}$. □

3.4.5. *Type preservation for* MICT

The type-preservation or subject-reduction property is not trivial to prove due to the absence of type annotations on terms characteristic of Curry-style systems. A detailled proof of this property will be given for the second type system presented in this article, proof which can easily be simplified to get a proof for the current system.

4. A Type System for Dialgebras

Here we develop the main contribution of this article, an extension of F with initial/final dialgebras as defined in section 2.3. This time we have multiple constructors/destructors, feature which simplifies programming and modularizes the definition of monotonicity witnesses involved in the typing rules.

4.1. Definition of the System

Extend system F as follows:

- Types:
$$A, B, C, F, G \ ::= \ \ldots \mid \mu X(F_1, \ldots, F_k) \mid \nu X(F_1, \ldots, F_k)$$
Here every F_i is called a *clause*.
- Terms :
$$t, r, s, m \quad ::= \quad \ldots \mid \mathsf{in}_{k,i}\, t \mid \mathsf{It}_k(\vec{m}, \vec{s}, t) \mid \mathsf{Rec}_k(\vec{m}, \vec{s}, t) \mid$$
$$\mathsf{CoIt}_k(\vec{m}, \vec{s}, t) \mid \mathsf{CoRec}_k(\vec{m}, \vec{s}, t) \mid \mathsf{out}_{k,i}\, t \mid \mathsf{out}_k^{-1}(\vec{m}, \vec{t}\,)$$
where in all cases the length of the vectors is k.
- Typing rules:
$$\frac{\Gamma \vdash t : F_i[X := \mu X(F_1, \ldots, F_k)]}{\Gamma \vdash \mathsf{in}_{k,i}\, t : \mu X(F_1, \ldots, F_k)} \ (\mu I)$$
$$\frac{\begin{array}{l}\Gamma \vdash t : \mu X(F_1, \ldots, F_k) \\ \Gamma \vdash m_i : F_i \,\mathsf{mon}\, X \ \ 1 \leq i \leq k \\ \Gamma \vdash s_i : F_i[X := B] \to B \ \ 1 \leq i \leq k\end{array}}{\Gamma \vdash \mathsf{It}_k(\vec{m}, \vec{s}, t) : B} \ (\mu E)$$
$$\frac{\begin{array}{l}\Gamma \vdash t : \mu X(F_1, \ldots, F_k) \\ \Gamma \vdash m_i : F_i \,\mathsf{mon}\, X \ \ 1 \leq i \leq k \\ \Gamma \vdash s_i : F_i[X := \mu X(F_1, \ldots, F_k) \times B] \to B \ \ 1 \leq i \leq k\end{array}}{\Gamma \vdash \mathsf{Rec}_k(\vec{m}, \vec{s}, t) : B} \ (\mu E^+)$$
$$\frac{\begin{array}{l}\Gamma \vdash s_i : B \to F_i[X := B] \ \ 1 \leq i \leq k \\ \Gamma \vdash m_i : F_i \,\mathsf{mon}\, X \ \ 1 \leq i \leq k \\ \Gamma \vdash t : B\end{array}}{\Gamma \vdash \mathsf{CoIt}_k(\vec{m}, \vec{s}, t) : \nu X(F_1, \ldots, F_k)} \ (\nu I)$$

$$\frac{\begin{array}{l}\Gamma \vdash s_i : B \to F_i[X := \nu X(F_1,\ldots,F_k) + B] \;\; 1 \le i \le k \\ \Gamma \vdash m_i : F_i \,\mathsf{mon}\, X \;\; 1 \le i \le k \\ \Gamma \vdash t : B\end{array}}{\Gamma \vdash \mathsf{CoRec}_k(\vec{m},\vec{s},t) : \nu X(F_1,\ldots,F_k)} \;(\nu I^+)$$

$$\frac{\begin{array}{l}\Gamma \vdash t_i : F_i[X := \nu X(F_1,\ldots,F_k)] \;\; 1 \le i \le k \\ \Gamma \vdash m_i : F_i \,\mathsf{mon}\, X \;\; 1 \le i \le k\end{array}}{\Gamma \vdash \mathsf{out}_k^{-1}(\vec{m},\vec{t}) : \nu X(F_1,\ldots,F_k)} \;(\nu I^i)$$

$$\frac{\Gamma \vdash r : \nu X(F_1,\ldots,F_k)}{\Gamma \vdash \mathsf{out}_{k,i}\, r : F_i[X := \nu X(F_1,\ldots,F_k)]} \;(\nu E)$$

- Operational semantics:

$$\begin{array}{rcl}
\mathsf{It}_k(\vec{m},\vec{s},\mathsf{in}_{k,i}\, t) & \mapsto_\beta & s_i\Big(m_i\big(\lambda x.\mathsf{It}_k(\vec{m},\vec{s},x)\big)t\Big) \\
\mathsf{Rec}_k(\vec{m},\vec{s},\mathsf{in}_{k,i}\, t) & \mapsto_\beta & s_i\Big(m_i\Big(\langle \mathsf{Id}, \lambda z.\mathsf{Rec}_k(\vec{m},\vec{s},z)\rangle\Big)t\Big) \\
\mathsf{out}_{k,i}\,\mathsf{CoIt}_k(\vec{m},\vec{s},t) & \mapsto_\beta & m_i\Big(\lambda z.\mathsf{CoIt}_k(\vec{m},\vec{s},z)\Big)(s_i t) \\
\mathsf{out}_{k,i}\,\mathsf{CoRec}_k(\vec{m},\vec{s},t) & \mapsto_\beta & m_i\Big([\mathsf{Id}, \lambda z.\mathsf{CoRec}_k(\vec{m},\vec{s},z)]\Big)(s_i t) \\
\mathsf{out}_{k,i}\,\mathsf{out}_k^{-1}(\vec{m},\vec{t}) & \mapsto_\beta & m_i(\lambda z.z)t_i
\end{array}$$

This finish the definition of the system MCICT. A system of Monotone and Clausular Inductive and Coinductive Types.

4.2. On the inverse for in

Proposition 1 on page 3 provides us with an inverse function ${\mathsf{in}_T}^{-1}$ which has no counterpart in the type system MICT. This inversion principle was left out of MICT only to keep an exact correspondence with the system MCICT where such inverse cannot be modelled in a direct way. However the principle can be added to MICT by adding a new term constructor $\mathsf{in}^{-1}(\cdot,\cdot)$ ruled by:

$$\frac{\begin{array}{c}\Gamma \vdash t : \mu X F \\ \Gamma \vdash m : F \,\mathsf{mon}\, X\end{array}}{\Gamma \vdash \mathsf{in}^{-1}(m,t) : F[X := \mu X F]}$$

with operational semantics $\mathsf{in}^{-1}(m, \mathsf{in}\, t) \mapsto_\beta m(\lambda z.z)t$.

With respect to dialgebras, equation (12) on page 6 is not suitable to be represented directly in our framework. The reason is that there is no satisfactory way to represent the tuples of objects $\langle F_1\mu,\ldots,F_k\mu\rangle$ Observe that the inverse given in page 6 is a function $\mathsf{in}_k^{-1} : \langle \mu,\ldots,\mu\rangle \to \langle F_1\mu,\ldots,F_k\mu\rangle$ such that

$$\mathsf{in}_k^{-1} \circ \langle \mathsf{in}_{k,1},\ldots,\mathsf{in}_{k,k}\rangle = \mathsf{Id}_{\langle F_1\mu,\ldots,F_k\mu\rangle}$$

So that we would need a rule like this:

$$\frac{\Gamma \vdash t : \big\langle \mu X(F_1,\ldots,F_k),\ldots,\mu X(F_1,\ldots,F_k)\big\rangle \quad \Gamma \vdash m_i : F_i \,\mathsf{mon}\, X \; 1 \le i \le k}{\Gamma \vdash \mathsf{in}_k^{-1}(\vec{m},t) : \Big\langle F_1[X := \mu X(F_1,\ldots,F_k)],\ldots,F_k[X := \mu X(F_1,\ldots,F_k)]\Big\rangle}$$

Of course we would need to give sense to a tuple of objects as a type, but this would complicate the system only to be able to model this principle.

On the other hand the main application of such rule is to define inductive destructors following the reasoning "If we have an inductive object $t : \mu X(F_1,\ldots,F_k)$ then it was generated by a clause $\mathsf{in}_k^{-1}\, t : F_i[X := \mu X(F_1,\ldots,F_k)]$ for some $1 \le i \le k$". This rule is used for instance, to guarantee that if t is a natural number then t is either 0 or a succesor $\mathsf{suc}\, n$. Such reasoning corresponds to an inverse $\mathsf{in}_k^{-1} : \mu \to F_1\mu + \ldots + F_k\mu$ such that $\mathsf{in}_k^{-1}(\vec{m}, \mathsf{in}_{k,i}\, t) = \mathsf{inj}_i^k\big(m_i(\lambda z.z)t\big)$, where inj_i^k is the canonical ith-injection.

We can model this kind of inverse by the rule:

$$\frac{\Gamma \vdash t : \mu X(F_1, \ldots, F_k) \quad \Gamma \vdash m_i : F_i \,\mathsf{mon}\, X \; 1 \leq i \leq k}{\Gamma \vdash \mathsf{in}_k^{-1}(\vec{m}, t) : F_1[X := \mu X(F_1, \ldots, F_k)] + \ldots + F_k[X := \mu X(F_1, \ldots, F_k)]} \; (\mu E^i)$$

But the main application of this rule, namely to define destructors on inductive types, can easily be achieved using primitive recursion, which is present in both of our systems. Therefore we will omit the rule as it would cause more problems than profits. One of its main disadvantages being the generation of a term inhabiting a sum type in an unusual way.

4.3. Programming in MCICT

Analogously to the previous system we can program functions with an inductive domain or coinductive codomain. In the case of a function $g : \mu X(F_1, \ldots, F_k) \to A$, the iteration principle ensures the existence of a program for g if g is defined by the following recurrence equations:

$$\begin{array}{rcl} g(\mathsf{in}_{k,1}\, x) & = & s_1\,(m_1\, g\, x) \\ & \vdots & \\ g(\mathsf{in}_{k,k}\, x) & = & s_k\,(m_k\, g\, x) \end{array}$$

where $s_i : F_i[X := A] \to A$ and $m_i : F_i \,\mathsf{mon}\, X, 1 \leq i \leq k$ are the fixed monotonicity witnesses used to eliminate the type $\mu X(F_1, \ldots, F_k)$. If these conditions hold, then the categorical machinery says that we can define $g := \lambda z.\mathsf{It}_k(\vec{m}, \vec{s}, z)$ and we will obtain the desired reduction behaviour:

$$g(\mathsf{in}_{k,i}\, x) \to_\beta^+ s_i\,(m_i\, g\, x)$$

Analogously primitive recursion provides a mean to program functions $g : \mu X(F_1, \ldots, F_k) \to A$ which satisfy the following recurrence equations:

$$\begin{array}{rcl} g(\mathsf{in}_{k,1}\, x) & = & s_1\,(m_1\, \langle \mathsf{Id}, g\rangle\, x) \\ & \vdots & \\ g(\mathsf{in}_{k,k}\, x) & = & s_k\,((m_k\, \langle \mathsf{Id}, g\rangle\, x) \end{array}$$

with $s_i : F_i[X := \mu X(F_1, \ldots, F_k) \times A] \to A$. In this case g can be defined as $g := \lambda z.\mathsf{Rec}_k(\vec{m}, \vec{s}, z)$.

In a dual way we can program a function $g : A \to \nu X(F_1, \ldots, F_k)$ which satisfy the following coiteration equations:

$$\begin{array}{rcl} \mathsf{out}_{k,1}(gx) & = & (m_1\; g)\;(s_1\; x) \\ & \vdots & \\ \mathsf{out}_{k,k}(gx) & = & (m_k\; g)\;(s_k\; x) \end{array}$$

where $s_i : A \to F_i[X := A]$ and $m_i : F_i \,\mathsf{mon}\, X, 1 \leq i \leq k$ are the fixed monotonicity witnesses used to introduce the type $\nu X(F_1, \ldots, F_k)$. In this case the program is $g := \lambda z.\mathsf{Colt}_k(\vec{m}, \vec{s}, z)$

Finally corecursion provides a mean to program functions $g : A \to \nu X(F_1, \ldots, F_k)$ which satisfy the following equations:

$$\begin{array}{rcl} \mathsf{out}_{k,1}(g\; x) & = & (m_1\; [\mathsf{Id}, g])\;(s_1\; x) \\ & \vdots & \\ \mathsf{out}_{k,k}(g\; x) & = & (m_k\; [\mathsf{Id}, g])\;(s_k\; x) \end{array}$$

with $s_i : A \to F_i[X := \nu X(F_1, \ldots, F_k) + A]$. In this case a program is $g := \lambda z.\mathsf{CoRec}_k(\vec{m}, \vec{s}, z)$

Next we show some examples of programming in MCICT. The reader is invited to program more functions and to verify the soundness of programs with respect to the operational semantics.

Example 9 (Unit and Empty Types)**.** *The two degenerated types with no clauses are the empty type* $\mathsf{void} := \mu X()$ *which cannot be inhabited and the unit type* $1 := \nu X()$ *which unique element will be denoted, as before, with* $\star$*.*

Example 10 (The Booleans)**.** *This very simple but useful type can be defined as* $\mathsf{bool} := \mu X(1,1)$

- *Canonical monotonicity witnesses:* $\mathsf{map}_1 := \mathsf{map}_2 := \lambda f \lambda x x$
- *Constructors:* $\mathsf{true} := \mathsf{in}_{2,1} \star,\ \mathsf{false} := \mathsf{in}_{2,2} \star$
- *Given a type A we define the conditional function*

$$\mathsf{if\_then\_else\_} : \mathsf{bool} \to A \to A \to A$$

with the following behaviour, for $r, s : A$:

$$\mathsf{if\ true\ then}\ r\ \mathsf{else}\ s = r$$
$$\mathsf{if\ false\ then}\ r\ \mathsf{else}\ s = s$$

This is easily defined by iteration as:

$$\mathsf{if\ \_\ then\ \_\ else\ \_} := \lambda z \lambda x \lambda y.\mathsf{It}_2(\mathsf{map}_1, \mathsf{map}_2, \lambda u.x, \lambda v.y, z)$$

where $u \neq x, v \neq y$ and $\mathsf{map}_1 := \mathsf{map}_2 := \lambda f \lambda x x$*.*

Example 11 (The natural numbers)**.** *Define* $\mathsf{nat} := \mu X(1, X)$

- *Canonical monotonicity witnesses:* $\mathsf{map}_1 := \lambda f \lambda x.x,\ \mathsf{map}_2 := \lambda x x$
- *Constructors:*
  - *Zero:* $0 : \mathsf{nat},\ 0 := \mathsf{in}_{2,1} \star$
  - *Succesor function:* $\mathsf{suc} : \mathsf{nat} \to \mathsf{nat},\ \mathsf{suc} := \mathsf{in}_{2,2}$
- *Destructors:*
  - *Predeccesor function:* $\mathsf{pred} : \mathsf{nat} \to 1 + \mathsf{nat}$, *such that*
    $\mathsf{pred}\, 0 = \mathsf{error},\ \mathsf{pred}\ \mathsf{suc}\, n = \mathsf{inr}\, n$
    $\mathsf{pred} := \lambda n.\mathsf{Rec}_2(\mathsf{map}_1, \mathsf{map}_2, \lambda y.\mathsf{case}(y, u.\,\mathsf{inl}\, u, v.\,\mathsf{inr}(\mathsf{fst}\, v)), n)$
- *Some functions on* nat*:*
  - $\mathsf{sum} : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat},\ \ \mathsf{sum} := \lambda n \lambda m.\mathsf{It}_2(\mathsf{map}_1, \mathsf{map}_2, \lambda x.n, \mathsf{suc}, m).$
  - $\mathsf{prod} : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}$
    $\mathsf{prod} := \lambda n \lambda m.\mathsf{It}_2(\mathsf{map}_1, \mathsf{map}_2, \mathsf{in}_{2,1}, \lambda y.\mathsf{sum}\, y\, n, m)$

Example 12 (Finite lists over A)**.** *Define* $\mathsf{list}(A) := \mu X(1, A \times X)$

- *Canonical monotonicity witnesses:* $\mathsf{map}_1 := \lambda f \lambda x x,\ \ \mathsf{map}_2 := \lambda f \lambda x.\langle \mathsf{fst}\, x, f(\mathsf{snd}\, x)\rangle$
- *Constructors:*
  - *Empty list:* $\mathsf{nil} : \mathsf{list}(A),\quad \mathsf{nil} := \mathsf{in}_{2,1} \star$
  - *Cons function:* $\mathsf{cons} : A \times \mathsf{list}(A) \to \mathsf{list}(A),\quad \mathsf{cons} := \mathsf{in}_{2,2}$
- *Destructors:*
  - *Head function:* $\mathsf{head} : \mathsf{list}(A) \to 1 + A$ *such that* $\mathsf{head}\, \mathsf{nil} = \mathsf{error},\ \mathsf{head}(\mathsf{cons}\langle a, \ell\rangle) = \mathsf{inr}\, a$
    $\mathsf{head} := \lambda z.\mathsf{Rec}(\mathsf{map}_1, \mathsf{map}_2, \lambda x.\,\mathsf{inl}\, x, \lambda y.\,\mathsf{inr}(\mathsf{fst}\, y), z)$
  - *Tail function:* $\mathsf{tail} : \mathsf{list}(A) \to 1 + \mathsf{list}(A)$ *such that* $\mathsf{tail}\, \mathsf{nil} = \mathsf{error},\ \mathsf{tail}(\mathsf{cons}\langle a, \ell\rangle) = \mathsf{inr}\, \ell$
    $\mathsf{tail} := \lambda z.\mathsf{Rec}(\mathsf{map}_1, \mathsf{map}_2, \lambda x.\,\mathsf{inl}\, x, \lambda y.\,\mathsf{inr}(\mathsf{fst}(\mathsf{snd}\, y))), z)$
- *Some funcions on* $\mathsf{list}(A)$*:*
  - *Append:* $\mathsf{app} : \mathsf{list}(A) \to \mathsf{list}(A) \to \mathsf{list}(A)$
    $\mathsf{app} := \lambda x.\mathsf{It}_2(\mathsf{map}_1, \mathsf{map}_2, \lambda y \lambda z z, \lambda u \lambda v.\,\mathsf{cons}\langle \mathsf{fst}\, u, (\mathsf{snd}\, u) v\rangle)$
  - *Length:* $\mathsf{length} : \mathsf{list}(A) \to \mathsf{nat}$
    $\mathsf{length} := \lambda x.\mathsf{It}_2(\mathsf{map}_1, \mathsf{map}_2, \lambda x.0, \lambda y.\,\mathsf{suc}(\mathsf{snd}\, y), x)$
  - *Reverse:* $\mathsf{rev} : \mathsf{list}(A) \to \mathsf{list}(A)$
    $\mathsf{rev} := \lambda x.\mathsf{It}_2(\mathsf{map}_1, \mathsf{map}_2, \mathsf{nil}, \lambda z.\mathsf{app}\ (\mathsf{snd}\, z)\ (\mathsf{cons}\langle \mathsf{fst}\, z, \mathsf{nil}\rangle), x)$
  - *The polymorphic map function on lists:* $\mathsf{maplist} : \forall X \forall Y.(X \to Y) \to \mathsf{list}(X) \to \mathsf{list}(Y)$ *such that:*
    $$\mathsf{maplist}\, f\, \mathsf{nil} = \mathsf{nil} \qquad \mathsf{maplist}\, f\, (x : xs) = (f\, x) : (\mathsf{maplist}\, f\, xs)$$
    where $x : xs$ denotes $\mathsf{cons}\langle x, xs\rangle$*. The program is:*
    $$\mathsf{maplist} = \lambda f \lambda x.\mathsf{It}_2(\mathsf{map}_1, \mathsf{map}_2, \lambda u.\mathsf{nil}, \lambda v.(f\ (\mathsf{fst}\, v)) : (\mathsf{snd}\, v), x)$$

Example 13 (Streams (infinite lists) over A)**.** $\mathsf{stream}(A) := \nu X(A, X)$

- *Canonical monotonicity witnesses:* $\mathsf{map}_1 := \lambda f \lambda x.x,\ \mathsf{map}_2 := \lambda f f$

- *Destructors:*
  - $\mathsf{head} : \mathsf{stream}(A) \to A, \ \ \mathsf{head} := \mathsf{out}_{2,1}$
  - $\mathsf{tail} : \mathsf{stream}(A) \to \mathsf{stream}(A), \ \ \mathsf{tail} := \mathsf{out}_{2,2}$
- *Constructor* $\mathsf{cons} : A \times \mathsf{stream}(A) \to \mathsf{stream}(A)$
  $\mathsf{cons} := \lambda x.\mathsf{CoRec}_2(\mathsf{map}_1, \mathsf{map}_2, \mathsf{fst}, \lambda z.\, \mathsf{inl}(\mathsf{snd}\, z), x)$
- *Some functions:*
  - *Streams of constants,* $\mathsf{cnt} : A \to \mathsf{stream}(A), \ \ \mathsf{head}(\mathsf{cnt}\, a) = a, \ \mathsf{tail}(\mathsf{cnt} a) = \mathsf{cnt}\, a$
    $\mathsf{cnt} := \lambda x.\mathsf{CoIt}_2(\mathsf{map}_1, \mathsf{map}_2, \lambda zz, \lambda zz, x)$
  - *The stream of natural numbers from a given one:*
    $\mathsf{from} : \mathsf{nat} \to \mathsf{stream}(\mathsf{nat}), \ \ \mathsf{head}(\mathsf{from}\, n) = n, \ \ \mathsf{tail}(\mathsf{from}\, n) = \mathsf{from}(\mathsf{suc}\, n)$
    $\mathsf{from} := \lambda x.\mathsf{CoIt}_2(\mathsf{map}_1, \mathsf{map}_2, \lambda zz, \mathsf{suc}, x)$
  - *The polymorphic map function on streams:*
    $\mathsf{mapstr} : \forall X \forall Y.(X \to Y) \to \mathsf{stream}(X) \to \mathsf{stream}(Y)$
    $\mathsf{head}(\mathsf{mapstr}\, f\, x) = f\, (\mathsf{head}\, x), \ \mathsf{tail}(\mathsf{mapstr}\, f\, x) = \mathsf{mapstr}\, f\, (\mathsf{tail}\, x)$
    $\mathsf{mapstr}\, f := \lambda z \mathsf{CoIt}(\mathsf{map}_1, \mathsf{map}_2, \lambda x.f\, (\mathsf{head}\, x), \ \mathsf{tail}, \ z)$

Example 14 (Infinite A-labelled binary trees)**.** $\mathsf{infbtree}\, A := \nu X(A, X \times X)$ *with destructors*

- $\mathsf{rlabel} : \mathsf{infbtree}\, A \to A, \ \ \mathsf{label} := \mathsf{out}_{2,1}$
- $\mathsf{children} : \mathsf{infbtree}\, A \to \mathsf{infbtree}\, A \times \mathsf{infbtree}\, A, \ \ \mathsf{children} := \mathsf{out}_{2,2}$

Observe that the second destructor encodes together the two subtrees of a given tree and therefore the use of some projections cannot be avoided.
A better definition would allow a direct destruction into the left and right subtrees as in the following example.

Example 15 (Finite and infinite A-labelled binary trees)**.**

$$\mathsf{FinInfBTree}\, A := \nu X(A, 1 + X, 1 + X)$$

with destructors label (of the root), left subtree and right subtree.

- $\mathsf{label} : \mathsf{infbtree}\, A \to A, \ \ \mathsf{label} := \mathsf{out}_{3,1}$
- $\mathsf{lst} : \mathsf{infbtree}\, A \to 1 + \mathsf{infbtree}\, A, \ \ \mathsf{lst} := \mathsf{out}_{3,2}$
- $\mathsf{rst} : \mathsf{infbtree}\, A \to 1 + \mathsf{infbtree}\, A, \ \ \mathsf{rst} := \mathsf{out}_{3,3}$

Example 16 (Finite branching, A-labelled trees with potentially infinite depth)**.** $\mathsf{pidtree}\, A := \nu X(A, \mathsf{list}(X))$ *with destructors*

- $\mathsf{rlabel} : \mathsf{pidtree}\, A \to A$
- $\mathsf{lstrees} : \mathsf{pidtree}\, A \to \mathsf{list}(\mathsf{pidtree}\, A)$

An inhabitant $t : \mathsf{pidtree}$ *is destructed as the label of its root* $\mathsf{rlabel}\, t$ *and the list of its immediate subtrees* $\mathsf{lstrees}\, t$*.*

- *Canonical monotonicity witnesses:* $\mathsf{map}_1 := \lambda f \lambda x x, \ \mathsf{map}_2 := \mathsf{maplist}$
  This example shows the important connection between the inductive and the coinductive part of our system. The coinductive subsystem depends on the inductive one to be able to define the monotonicity witness of a coinductive type. In this case the iteratively defined witness maplist *for the coinductive type* $\mathsf{pidtree}\, A$*.*
- *The function* $\mathsf{maptree} : (A \to C) \to \mathsf{pidtree}\, A \to \mathsf{pidtree}\, C$*, mapping a function* $f : A \to C$ *into a tree* $t : \mathsf{pidtree}\, A$ *is coiteratively defined by:*

$$\begin{aligned} \mathsf{rlabel}\, (\mathsf{maptree}\ f\, t) &= f\, (\mathsf{rlabel}\, t) \\ \mathsf{lstrees}\, (\mathsf{maptree}\ f\, t) &= \mathsf{maplist}\ \mathsf{maptree}\, (\mathsf{lstrees}\, t) \end{aligned}$$

A program for $\mathsf{maptree}\, f$ *is:*

$$\mathsf{maptree}\ f := \lambda x.\mathsf{CoIt}_2(\mathsf{map}_1, \ \mathsf{map}_2, \ \lambda y.f\, (\mathsf{rlabel}\, y), \ \mathsf{lstrees}, \ x)$$

Example 17 (Deterministic automata with input alphabet Σ and output type B)**.**

$$\mathsf{daut}(\Sigma, B) := \nu X(\Sigma \to X, B)$$

- *Canonical monotonicity witnesses:* $\mathsf{map}_1 := \lambda f \lambda g \lambda x.f(gx), \mathsf{map}_2 := \lambda f \lambda x x$

- *Destructors:*
  - $\mathsf{next} : \mathsf{daut}(\Sigma, B) \to (\Sigma \to \mathsf{daut}(\Sigma, B))$
  - $\mathsf{obs} : \mathsf{daut}(\Sigma, B) \to B$

This example provides the fundamentals of the coalgebraic automata theory developed in [21]. In this setting an automata is a pair $M = \langle \delta, o \rangle$ with $\delta : Q \to \Sigma \to Q$ the transition function and $o : Q \to B$ an observation function, thus M is a Moore automata. There is no need neither for the existence of an initial state nor for assuming that Q or Σ are finite.
The coinductive type $\mathsf{daut}(\Sigma, B)$ is inhabited essentially by behaviour functions $\mathsf{beh}(q) : \Sigma^\star \to B$ for a given state $q \in Q$.
We can codify such an automata by means of a function $\mathsf{caut} : Q \to \mathsf{daut}(\Sigma, B)$ defined coiteratively:

$$\mathsf{caut} := \lambda z.\mathsf{CoIt}_2(\mathsf{map}_1, \mathsf{map}_2, \delta, o, z)$$

destructed as follows:

$$\mathsf{next}\,\mathsf{caut}\,q = \lambda x.\mathsf{caut}((\delta q)x) \qquad \mathsf{obs}\,\mathsf{caut}\,q = oq$$

Given a behaviour function $\mathsf{beh}(q) : \Sigma^\star \to B$ we can codify it by an inhabitant of $\mathsf{daut}(\Sigma, B)$ by means of a function $\mathsf{cbeh} : (\Sigma^\star \to B) \to \mathsf{daut}(\Sigma, B)$ given by:

$$\mathsf{cbeh} := \lambda z.\mathsf{CoIt}_2(\mathsf{map}_1, \mathsf{map}_2, \lambda f \lambda a \lambda w.f(a.w), \lambda g.g(\varepsilon), z)$$

where $\varepsilon, a.w$ denote the nil and cons operations respectively on $\Sigma^\star$. This function is destructed as follows:

$$\mathsf{obs}(\mathsf{cbeh}\,beh(q)) = beh(q)(\varepsilon) \qquad \mathsf{next}(\mathsf{cbeh}\,beh(q)) = \lambda a.\mathsf{cbeh}\big(\lambda w.beh(q)(a.w)\big)$$

Other types for automata are:

- Partial automata: $\mathsf{paut}(\Sigma, B) := \nu X(\Sigma \to 1 + X, B)$.
  In this case the transition functions are of the form $\delta : Q \to \Sigma \to 1 + Q$ such that $\delta\, q\, a = \mathsf{error}$ if and only if such transition is undefined.
- Finite Deterministic Automata: Just take Q, Σ finite and $B = \mathsf{bool}$.

  $$\mathsf{fda}(\Sigma) := \mathsf{daut}(\Sigma, \mathsf{bool}) = \nu X(\Sigma \to X, \mathsf{bool})$$

  Some functions on this type are:
  - The complement of a FDA: $\mathsf{comp} : \mathsf{fda}(\Sigma) \to \mathsf{fda}(\Sigma)$ destructed as:

    $$\mathsf{next}\,(\mathsf{comp}\,M) = \mathsf{next}\,M \qquad \mathsf{obs}\,(\mathsf{comp}\,M) = \mathsf{not}\,(\mathsf{obs}\,M)$$

  - The product automata of two FDA's: $\mathsf{prod} : \mathsf{fda}(\Sigma) \to \mathsf{fda}(\Sigma) \to \mathsf{fda}(\Sigma)$ destructed as follows:

    $$\mathsf{next}\,(\mathsf{prod}\,M_1\,M_2) = \mathsf{prod}\,(\mathsf{next}\;M_1)\,(\mathsf{next}\;M_2)$$

    According to the language we want to recognize we have different possibilites for the observation function:

    $$\mathsf{obs}\,(\mathsf{prod}\,M_1\,M_2) = (\mathsf{obs}\,M_1)\,\mathsf{and}\,(\mathsf{obs}\,M_2) \;\text{ for }\; L(M_1) \cap L(M_2)$$

    $$\mathsf{obs}\,(\mathsf{prod}\,M_1\,M_2) = (\mathsf{obs}\,M_1)\,\mathsf{or}\,(\mathsf{obs}\,M_2) \;\text{ for }\; L(M_1) \cup L(M_2)$$

    $$\mathsf{obs}\,(\mathsf{prod}\,M_1\,M_2) = (\mathsf{obs}\,M_1)\,\mathsf{and\ not}\,(\mathsf{obs}\,M_2) \;\text{ for }\; L(M_1) - L(M_2)$$

Example 18 (Potentially infinite trees with A-labelled branches)**.**

$$\mathsf{BLTree}\,A := \nu X(\mathsf{list}(A \times X))$$

A tree $t : \mathsf{BLTree}\,A$ is destructed by $\mathsf{lsb} : \mathsf{BLTree}\,A \to \mathsf{list}(A \times \mathsf{BLTree}\,A)$, $\mathsf{lsb} := \mathsf{out}$, $\mathsf{lsb}\;t$ returns the list of branches pendant from the root of t, i.e., returns a list of pairs of $A \times \mathsf{BLTree}\,A$ consisting of a label $a : A$ for the branch and the list of subtrees pendant from that branch in order from left to right.

- *Canonical monotonicity witness: The canonical witness*

$$\mathsf{map} : \forall X \forall Y.(X \to Y) \to \mathsf{list}(A \times X) \to \mathsf{list}(A \times Y)$$

is defined as follows:

$$\mathsf{map}\, f\, \mathsf{nil} = \mathsf{nil} \qquad \mathsf{map}\, f\, (\langle a, x\rangle : xs) = \langle a, fx\rangle : (\mathsf{map}\, f\, xs)$$

But this is just a composition of the iteratively defined maplist *function with the function* $G : \forall X \forall Y.(X \to Y) \to (A \times X) \to (A \times Y)$ *such that* $G\, f\, \langle a, x\rangle = \langle a, f\, x\rangle$. *Again an inductive definition is required to define a monotonicity witness for a coinductive type.*
- *Breadth first search: we will program a function* $\mathsf{bfs} : \mathsf{BLTree}\, A \to A^\infty$, *where* A^∞ *is the type of finite and infinite lists over* A, *which takes a tree* t *and returns a list of the branches in breadth-first order. To do this we first define a function* $\mathsf{bfl} : \mathsf{list}(A \times \mathsf{BLTree}\, A) \to A^\infty$ *and set* $\mathsf{bfs} := \mathsf{bfl} \circ \mathsf{lsb}$. *The coinductive definition is:*

$$\mathsf{head}\, \mathsf{bfl}\, t = \mathsf{if}\ (\mathsf{isnil}\, (\mathsf{bfl}\, t))\ \mathsf{then\ error\ else}\ \mathsf{inr}(\mathsf{fst}(\mathsf{head}\, t))$$

$$\mathsf{tail}\, \mathsf{bfl}\, t = \mathsf{if}\ (\mathsf{isnil}\, (\mathsf{bfl}\, t))\ \mathsf{then\ error\ else}\ \mathsf{inr}\, \mathsf{bfl}((\mathsf{tail}\, t) * \mathsf{lsb}(\mathsf{snd}(\mathsf{head}\, t)))$$

where $*$ *denotes append of lists.*
bfl *is programmed by coiteration as*

$$\mathsf{bfl} := \lambda x.\mathsf{CoIt}_2(\mathsf{map}_1, \mathsf{map}_2, s_1, s_2, x)$$

where
  - $\mathsf{map}_1 := \lambda f \lambda x x, \ \ \mathsf{map}_2 := \lambda f \lambda x.\mathsf{case}(x, y.\, \mathsf{inl}\, y, z.\, \mathsf{inr}\, f z)$
  - $s_1 : \mathsf{list}(A \times \mathsf{BLTree}\, A) \to 1 + A$

$$s_1 := \lambda w.\mathsf{if}\ (\mathsf{isnil}\, w)\ \mathsf{then\ error\ else}\ \mathsf{inr}(\mathsf{fst}(\mathsf{head}\, w))$$

  - $s_2 : \mathsf{list}(A \times \mathsf{BLTree}\, A) \to 1 + \mathsf{list}(A \times \mathsf{BLTree}\, A)$

$$s_2 := \lambda w.\mathsf{if}\ (\mathsf{isnil}\, w)\ \mathsf{then\ error\ else}\ \mathsf{inr}(\mathsf{tail}\, w * \mathsf{lsb}(\mathsf{snd}(\mathsf{head}\, w)))$$

- *Depth first search: this can be easily obtained from the above program by changing the second step function to:*
  $s_2 : \mathsf{list}(A \times \mathsf{BLTree}\, A) \to 1 + \mathsf{list}(A \times \mathsf{BLTree}\, A)$

$$s_2 := \lambda w.\mathsf{if}\ (\mathsf{isnil}\, w)\ \mathsf{then\ error\ else}\ \mathsf{inr}(\mathsf{lsb}(\mathsf{snd}(\mathsf{head}\, w)) * (\mathsf{tail}\, w))$$

The above examples show the advantage of using clauses. There is a direct definition of constructors or destructors which avoids the use of injections or projections in general. This feature also modularizes the definitons of monotonicity witnesses and the general mechanism of (co)inductive definitions of functions.

4.4. MCICT IS SAFE

We prove termination of the system by embedding it into the already terminating system MICT. For the type preservation we give here a direct proof.

4.4.1. *Strong Normalization of* MCICT

Strong normalization of the clausular system MCICT will follow the standard technique of type-respecting reduction-preserving translations or embeddings, see [12]. This time an embedding $(\cdot)'$, into the system MICT will be given. The main idea is to define $\mu X(A_1, \ldots, A_k)'$ as $\mu X.A'_1 + \ldots + A'_k$ and $\nu X(A_1, \ldots, A_k)'$ as $\nu X.A'_1 \times \ldots \times A'_k$. Some details are given below.

From now on we agree to associate sum and product to the right.

Definition 20. *The following syntactic sugar will be useful, where $k \geq 2$: we fix $k \geq 2$.*

$$\begin{array}{rcl} \mathsf{inj}_j^k & := & \lambda z.\, \mathsf{inr}^{j-1}(\mathsf{inl}\, z), \quad 1 \leq j < k \\ \mathsf{inj}_k^k & := & \lambda z.\, \mathsf{inr}^{k-1}\, z \\ \pi_{k,j} & := & \lambda z.\, \mathsf{fst}(\mathsf{snd}^{j-1}\, z), \quad 1 \leq j < k \\ \pi_{k,k} & := & \lambda z.\, \mathsf{snd}^{k-1}\, z \end{array}$$

These are, of course, the canonical injections and projections for a k-sum and k-product.

Definition 21 (MICT)**.** *Given variables $x_1, \ldots, x_k, y_1, \ldots, y_k, f, u, v, w, z$ we define, for $k \geq 2$ and $1 \leq i \leq k$, the following families of terms t_i, r_i, q_i, p_i:*

$$\begin{array}{rcll} t_j[u] & := & \mathsf{inj}_j^k(x_j f u) & 1 \leq j \leq k \\ \\ r_0[v] & := & t_k[v] & \\ r_{j+1}[v] & := & \mathsf{case}(v, x.t_{k-(j+1)}[x], y.r_j[y]) & 0 \leq j < k-1 \\ \\ q_0[w] & := & y_k w & \\ q_{j+1}[w] & := & \mathsf{case}(w, x.y_{k-(j+1)}x, y.q_j[y]) & 0 \leq j < k-1 \\ \\ p_j[z] & := & x_j f(\pi_{k,j} z) & 1 \leq j \leq k \end{array}$$

Observe that free variables are:

$$\begin{array}{rcl} FV(t_i[u]) & = & \{x_i, f, u\} \\ FV(r_i[v]) & = & \{x_{k-i}, \ldots, x_k, f, v\} \\ FV(q_i[w]) & = & \{y_{k-i}, \ldots, y_k, w\} \\ FV(p_i[z]) & = & \{x_i, f, z\} \end{array}$$

Definition 22. *Given variables $\vec{x}, \vec{y}$ with $|\vec{x}| = |\vec{y}| = k$ define the following terms:*

$$\begin{array}{rcl} \mathbb{M}^+[\vec{x}] & := & \lambda f \lambda z.r_{k-1}[z] \\ \mathbb{S}^+[\vec{y}] & := & \lambda w.q_{k-1}[w] \\ \mathbb{M}^\times[\vec{x}] & := & \lambda f.\lambda z.\langle p_1[z], \ldots, p_k[z]\rangle \\ \mathbb{S}^\times[\vec{y}] & := & \lambda w.\langle y_1 w, \ldots, y_k w\rangle \end{array}$$

Observe that

$$FV(\mathbb{M}^+[\vec{x}]) = FV(\mathbb{M}^\times[\vec{x}]) = \vec{x}$$
$$FV(\mathbb{S}^+[\vec{y}]) = FV(\mathbb{S}^\times[\vec{x}]) = \vec{y}$$

These terms will be needed for the embedding of (co)iterators, (co)recursors and $\mathsf{in}\,/\,\mathsf{out}$ functions.

Definition 23. *The embedding $(\cdot)' : \mathsf{MCICT} \to \mathsf{MICT}$ is defined in two parts, first we define it for the degenerate cases of (co)inductive types without arguments, that is, $\mu X(), \nu X()$, which are special encoded types. Then we give the general definition which excludes the previous cases. The ommited cases are just homomorphic*[2]

$$\begin{array}{rcl} (\mu X())' & := & \forall X X \\ \mathsf{It}_0(t)' & := & t' \\ \mathsf{Rec}_0(t)' & := & t' \\ (\nu X())' & := & \forall X.X \to X \\ \mathsf{CoIt}_0(t)' & := & \lambda z z \\ \mathsf{CoRec}_0(t)' & := & \lambda z z \end{array}$$

[2] V.gr. $X' = X$, $(A \to B)' = A' \to B'$, $(\forall X A)' = \forall X.A'$, etc.

Next the general definition where $k \geq 1$

$$
\begin{array}{rcl}
\big(\mu X(F_1,\ldots,F_k)\big)' & := & \mu X.F_1' + \ldots + F_k' \\
\big(\nu X(F_1,\ldots,F_k)\big)' & := & \nu X.F_1' \times \ldots \times F_k' \\
x' & := & x \\
\mathsf{in}_{1,1}\, t' & := & \mathsf{in}\, t' \\
\mathsf{in}_{k,i}\, t' & := & \mathsf{in}(\mathsf{inj}_i^k\, t')\ k \geq 2 \\
\mathsf{It}_1(m,s,t)' & := & \mathsf{It}(m',s',t') \\
\mathsf{It}_k(\vec{m},\vec{s},t)' & := & \mathsf{It}(\mathbb{M}^+[\vec{m}'],\mathbb{S}^+[\vec{s}\,'],t') \quad k \geq 2 \\
\mathsf{Rec}_1(m,s,t)' & := & \mathsf{Rec}(m',s',t') \\
\mathsf{Rec}_k(\vec{m},\vec{s},t)' & := & \mathsf{Rec}(\mathbb{M}^+[\vec{m}'],\mathbb{S}^+[\vec{s}\,'],t') \quad k \geq 2 \\
(\mathsf{out}_{1,1}\, t)' & := & \mathsf{out}\, t' \\
(\mathsf{out}_{k,i}\, t)' & := & \pi_{k,i}(\mathsf{out}\, t')\ k \geq 2 \\
\mathsf{out}_k^{-1}(\vec{m},\vec{t}\,)' & := & \mathsf{out}^{-1}(\mathbb{M}^\times[\vec{m}'],\langle t_1',\ldots,t_k'\rangle) \\
\mathsf{CoIt}_1(m,s,t)' & := & \mathsf{CoIt}(m',s',t') \\
\mathsf{CoIt}_k(\vec{m},\vec{s},t)' & := & \mathsf{CoIt}(\mathbb{M}^\times[\vec{m}'],\mathbb{S}^\times[\vec{s}\,'],t') \quad k \geq 2 \\
\mathsf{CoRec}_1(m,s,t)' & := & \mathsf{CoRec}(m',s',t') \\
\mathsf{CoRec}_k(\vec{m},\vec{s},t)' & := & \mathsf{CoRec}(\mathbb{M}^\times[\vec{m}'],\mathbb{S}^\times[\vec{s}\,'],t') \quad k \geq 2
\end{array}
$$

where the terms $\mathbb{M}^+, \mathbb{M}^\times, \mathbb{S}^+, \mathbb{S}^\times$ are taken from definition 22.

The proofs of type-respect and reduction preservation are routinary.

Proposition 9 (Type respect)**.** *If $\Gamma \vdash_{\mathsf{MCICT}} r : B$ then $\Gamma' \vdash_{\mathsf{MICT}} r' : B'$, where if $\Gamma = \{x_1 : A_1, \ldots, x_n : A_n\}$ then $\Gamma' = \{x_1 : A_1', \ldots, x_n : A'x_n\}$.*

Proof. Induction on $\vdash_{\mathsf{MCICT}}$. □

Proposition 10 (Preservation of reduction)**.** *If $r \to_\beta s$ in* MCICT *then $r' \to_\beta^+ s'$ in* MICT*.*

Proof. Induction on $\to_\beta$ in MCICT. □

Proposition 11. MCICT *is strongly normalising.*

Proof. Immediate from propositions 8 and 10. □

4.5. Type preservation for MCICT

In this section we give a detailed proof of type preservation (subject reduction) for the system MCICT. This important property is not trivial to prove because of the absence of type anotations in terms given by the Curry-style presentation and due to the presence of untraceable typing rules, which are rules whose application cannot be traced by looking only at terms.

Definition 24. *Given a type A and a context Γ we define the set $\mathcal{C}_\Gamma(A)$ of Γ-instances of A as the least class of types such that:*

- *$A \in \mathcal{C}_\Gamma(A)$ $(I1)$*
- *If $B \in \mathcal{C}_\Gamma(A)$ and $X \notin FV(\Gamma)$ then $B[X := F] \in \mathcal{C}_\Gamma(A)$. $(I2)$.*

Lemma 18. *If $X \notin FV(\Gamma)$ then $\mathcal{C}_\Gamma(B[X := F]) \subseteq \mathcal{C}_\Gamma(B)$.*

Proof. By I1, $B \in \mathcal{C}_\Gamma(B)$ which implies, as $X \notin FV(\Gamma)$, that $B[X := \mathcal{F}] \in \mathcal{C}_\Gamma(B)$, the claim follows now by the minimality of $\mathcal{C}_\Gamma(B[X := F])$. □

Definition 25. *A formula A is an open formula if it is not an universal quantification. The interior of a formula A, denoted A° is defined as follows:*

$$
\begin{array}{rcll}
A^\circ & := & A, & \text{if } A \text{ is open.} \\
(\forall X A)^\circ & := & A^\circ &
\end{array}
$$

Definition 26. *We say that an inference rule is non-traceable if its application is not reflected in the type system. That is, if the term in the conclusion equals one of the terms of the premisses. Otherwise the rule is called traceable.*

In MCICT the non-traceable rules are the two rules for $\forall$

Lemma 19 (Main Lemma)**.** *Let $\widetilde{A}$ be an open formula. If $\Gamma \vdash t : \widetilde{A}$ is derived from $\Gamma \vdash t : A$ using only non-traceable rules then $\widetilde{A} \in \mathcal{C}_\Gamma(A^\circ)$*

Proof. Induction on the number of steps in the derivation of $\Gamma \vdash t : \widetilde{A}$ from $\Gamma \vdash t : A$. Case Analysis on the first rule used in that derivation.

- $(\forall I)$. We have $\Gamma \vdash t : \widetilde{A}$ from $\Gamma \vdash t : \forall X A$ where $X \notin FV(\Gamma)$, therefore by IH we get $\widetilde{A} \in \mathcal{C}_\Gamma((\forall X A)^\circ)$. But $(\forall X A)^\circ \equiv A^\circ$ therefore $\widetilde{A} \in \mathcal{C}_\Gamma(A^\circ)$.
- $(\forall E)$. We have $A \equiv \forall X \forall Y_1 \ldots \forall Y_n B$ with B open, i.e., $A^\circ = B = B^\circ$ and after the application of $(\forall E)$ we get $\Gamma \vdash t : B[X := F]$. By IH we have $\widetilde{A} \in \mathcal{C}_\Gamma\big((\forall Y_1 \ldots \forall Y_n . B[X := F])^\circ\big) = \mathcal{C}_\Gamma(B[X := F]^\circ)$. We have two subcases:

  - $B^\circ \neq X$. In this case $B[X := F]$ is open of the same form as B. Then the IH yields $\widetilde{A} \in \mathcal{C}_\Gamma(B[X := F])$ which implies by lemma 18, as w.l.o.g. $X \notin FV(\Gamma)$, that $\widetilde{A} \in \mathcal{C}_\Gamma(B)$ i.e.,$\widetilde{A} \in \mathcal{C}_\Gamma(A^\circ)$

  - $B^\circ \equiv X$. In this case $B[X := F] = F$ and the IH yields $\widetilde{A} \in \mathcal{C}_\Gamma(B[X := F]^\circ) = \mathcal{C}_\Gamma(F^\circ) = \mathcal{C}_\Gamma(B[X := F^\circ])$,which implies by lemma 18, as w.l.o.g. $X \notin FV(\Gamma)$, that $\widetilde{A} \in \mathcal{C}_\Gamma(B)$ i.e.,$\widetilde{A} \in \mathcal{C}_\Gamma(A^\circ)$

□

Definition 27. *A term t is called an I-term if it was generated by an introduction rule, i.e., I-terms are terms of the following shapes:*

$$\lambda x r,\ \langle r, s\rangle,\ \mathsf{inl}\, r,\ \mathsf{inr}\, s, \mathsf{in}_{k,j}\, r,\ \mathsf{CoIt}_k(\vec{m}, \vec{s}, r),\ \mathsf{CoRec}_k(\vec{m}, \vec{s}, r),\ \mathsf{out}_k^{-1}(\vec{m}, \vec{r})$$

Analogously E-terms are terms generated by an elimination rule, i.e. are terms of the following shapes:

$$r s,\ \mathsf{fst}\, r,\ \mathsf{snd}\, r,\ \mathsf{case}(r, x.s, y.t),\ \mathsf{It}_k(\vec{m}, \vec{s}, r),\ \mathsf{Rec}_k(\vec{m}, \vec{s}, r),\ \mathsf{out}_{k,j}\, r$$

Lemma 20 (Generation Lemma)**.** *If $\Gamma \vdash t : A$, where A is an open formula then:*

- *If t is the variable x then there exists a declaration $x : B \in \Gamma$ such that $A \in \mathcal{C}_\Gamma(B^\circ)$.*
- *If t is an I-term then $\Gamma \vdash t : A$ is the conclusion of an instance of the rule generating t.*
- *if t is an E-term then there exists a formula B such that $\Gamma \vdash t : B$ is the conclusion of the rule generating t and $A \in \mathcal{C}_\Gamma(B^\circ)$.*

Proof. Consider in the derivation $\Gamma \vdash t : A$ the last step where a traceable rule $\mathcal{R}$ occurs, thus $\mathcal{R}$ is the rule generating t. Suppose that the conclusion of $\mathcal{R}$ is $\Gamma \vdash t : B$. The main lemma implies that $A \in \mathcal{C}_\Gamma(B^\circ)$. Case Analysis on t.

- $t \equiv x$. Then $\mathcal{R}$ is (Var) and therefore exists $x : B \in \Gamma$ and as mentioned before $A \in \mathcal{C}_\Gamma(B^\circ)$.
- t is an E-term. This case is immediate as $\mathcal{R}$ is the rule generating t.
- t is an I-term. Case analysis on the shape of t. We concentrate on $t \equiv \mathsf{in}_{k,j}\, r$. In this case $\mathcal{R}$ is (μI), $B \equiv \mu Y(C_1, \ldots, C_\ell)$ and $\Gamma \vdash r : C_j[Y := \mu Y(C_1, \ldots, C_\ell)]$. Clearly $B = B^\circ$, therefore $A \in \mathcal{C}_\Gamma(B)$. Let

$$\mathcal{C} = \big\{\, \mu X(D_1, \ldots, D_k) \mid \Gamma \vdash r : D_j[X := \mu X(D_1, \ldots, D_k)], \text{ for some } k, D_i \,\big\},$$

we need to show that $A \in \mathcal{C}$. We claim that $\mathcal{C}_\Gamma(B) \subseteq \mathcal{C}$.
$(I1)$ Obviously $B \in \mathcal{C}$.
$(I2)$ Assume $R \in \mathcal{C}$ and $Z \notin FV(\Gamma)$. We have

$$R[Z := F] \equiv \mu X(D_1, \ldots, D_k)[Z := F] \equiv \mu X\big(D_1[Z := F], \ldots, D_k[Z := K]\big).$$

$R \in \mathcal{C}$ implies $\Gamma \vdash r : D_j[X := \mu X(D_1, \ldots, D_k)]$. From this, as $Z \notin FV(\Gamma)$ we can build a derivation of

$$\Gamma \vdash r : D_j[X := \mu X(D_1, \ldots, D_k)][Z := F].$$

Finally using substitution properties we obtain

$$\Gamma \vdash r : D_j[Z := F]\big[X := \mu X(D_1, \ldots, D_k)[Z := F]\big],$$

i.e.

$$\Gamma \vdash r : D_j[Z := F]\big[X := \mu X(D_1[Z := F], \ldots, D_k[Z := F])\big]$$

which implies $R[Z := F] \in \mathcal{C}$.

Therefore by minimality of $\mathcal{C}_\Gamma(B)$ we conclude $\mathcal{C}_\Gamma(B) \subseteq \mathcal{C}$ which yields $A \in \mathcal{C}$. □

Proposition 12 (One-step Subject Reduction)**.** *If $\Gamma \vdash t : A$ and $t \to^1_\beta \widehat{t}$ (i.e. $t \to_\beta \widehat{t}$ in one step) then $\Gamma \vdash \widehat{t} : A$.*

Proof. Induction on $\vdash$. The cases for an introduction rule are direct from the IH. The cases for elimination rules are direct, as example we show it for (νE).

We have $A \equiv F_i[X := \nu X(F_1, \ldots, F_k)]$ and $\Gamma \vdash \mathsf{out}_{k,j}\, s : A$ coming from $\Gamma \vdash s : \nu X(F_1, \ldots, F_k)$. The interesting subcases are $s \equiv \mathsf{Colt}_k(\vec{m}, \vec{s}, r), \mathsf{CoRec}_k(\vec{m}, \vec{s}, r), \mathsf{out}^{-1}(\vec{m}, \vec{r})$. We analyze the case $s \equiv \mathsf{Colt}_k(\vec{m}, \vec{s}, r)$ and $\widehat{t} = m_i\big(\lambda z.\mathsf{Colt}_k(\vec{m}, \vec{s}, z)\big)(s_i r)$. From the assumption $\Gamma \vdash \mathsf{Colt}_k(\vec{m}, \vec{s}, r) : \nu X(F_1, \ldots, F_k)$, the generation lemma yields $\Gamma \vdash m_i : F_i \,\mathsf{mon}\, X$, $\Gamma \vdash s_i : B \to F_i[X := B]$, $\Gamma \vdash r : B$. It is easy to see that $\Gamma \vdash \lambda z.\mathsf{Colt}_k(\vec{m}, \vec{s}, z) : B \to \nu X(F_1, \ldots, F_k)$, which implies $\Gamma \vdash m_i\big(\lambda z.\mathsf{Colt}_k(\vec{m}, \vec{s}, z)\big) : F_i[X := B] \to F_i[X := \nu X(F_1, \ldots, F_k)]$.
On the other hand we have $\Gamma \vdash s_i r : F_i[X := B]$. Therefore $\Gamma \vdash m_i\big(\lambda z.\mathsf{Colt}_k(\vec{m}, \vec{s}, z)\big)(s_i r) : F_i[X := \nu X(F_1, \ldots, F_k)]$, i.e. $\Gamma \vdash \widehat{t} : A$. □

Corollary 10 (Subject Reduction for MCICT)**.** *If $\Gamma \vdash_{\mathbb{E}} r : A$ and $r \to_\beta \widehat{r}$ then $\Gamma \vdash_{\mathbb{E}} \widehat{r} : A$.*

Proof. Induction on the length of the reduction sequence $r \to_\beta \widehat{r}$. □

5. Conclusions

We have presented two Curry-style extensions of system F with monotone (co)inductive types including not only (co)iteration but also primitive (co)recursion principles and coinductive inversion. Both systems are safe and therefore suitable to be implemented as functional programming languages, especially the second one which allows the use of several constructors/destructors and is therefore more friendly to the user as it modularizes definitions of functions and monotonicity witnesses as was showed in several examples.

5.1. Related Work

Systems MCICT developed here can be seen as an extension of Hagino's categorical type system ([7]) with polymorphism and primitive (co)recursion. On the other hand one of the higher-order systems of [1] can be seen as an extension of the (co)iterative fragment of our first system MICT. Some related extensions in Church-style are studied in [5, 12].

Our systems have also a counterpart in natural deduction under the Curry-Howard correspondence developed by extending the second-order logic $AF2$ in [18], systems which are also related to the work in [23], and which are useful to extract programs from proofs à la Krivine-Parigot (see [19]), using as the underlying programming language the formalisms of this article. This has been done again in [18] where we have also developed Mendler-style sistems of (co)inductive types. The expressive power of such systems as well as its importance in higher-order functional programming are well-known (see [1], for example).

5.2. Future Work

It is desirable to implement our type systems, an starting point in this direction are the implementations in Haskell given in [24]. On the theoretical side we are currently working on some extensions of system F with course-of-value (co)iteration and (co)-recursion principles related to some systems developed in [23], the main goal is to define type systems modelling not only histomorphisms but also hylomorphisms. The categorical counterpart has already been developed in [2,22,24]. However the course-of-value-system in [23] does not correspond to the categorical interpretation of course-of-value induction, that is, the operational semantics does not correspond to the universal property of histomorphisms. It will be interesting to investigate in which sense both reduction concepts are related.

References

[1] A. Abel, R. Matthes, T. Uustalu. Iteration and Coiteration Schemes for Higher-Order and Nested Datatypes. In *Theoretical Computer Science* 333(1-2). pp. 3-66. Elsevier 2005.

[2] Alcino Cunha M. Recursion Patterns as Hylomorphisms. Technical Report DI-PURe-03.11.01, Department of Informatics, University of Minho. November 2003.

[3] R.L. Crole. Categories for Types. Cambridge Mathematical Textbooks. Cambridge University Press, 1993.

[4] H. Dybkjær, A. Melton. Comparing Hagino's Categorical Programming Language and Typed Lambda-Calculi.*Theoretical Computer Science* **111** pp. 145-189. Elsevier 1991.

[5] H. Geuvers. Inductive and coinductive types with iteration and recursion. In B. Nordström, K. Petterson, G. Plotkin, Eds. *Proceedings of the 1992 Workshop on Types for Proofs and Programs* Båstad, Sweden June 1992, pp. 183-207. Available Via `http://www.cs.kun.nl/~herman/BRABasInf_RecTyp.ps.gz`.

[6] J. Greiner. Programming with Inductive and Co-Inductive Types. Technical Report CMU-CS-92-109, Carnegie-Mellon University. January 1992

[7] T. Hagino. A Typed Lambda Calculus with Categorical Type Constructors. In D.H. Pitt, A. Poigné, D.E. Rydeheard. *Category Theory and Computer Science*. LNCS **283** Springer Verlag 1987.

[8] T. Hagino. A Categorical Programming Language. Ph.D. Thesis CST-47-87 (also published as ECS-LFCS-87-38). Department of Computer Science, University of Edinburgh 1987.

[9] B. Jacobs, J. Rutten. A Tutorial on (Co)Algebras and (Co)Induction. EATCS Bulletin 62. p. 222-259. 1997.

[10] S. Mac Lane. Categories for the Working Mathematicioan. 2nd. Edition. Vol. 5. Graduate Texts in Mathematics, Springer Verlag 1998.

[11] Ralph Matthes, Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types, Dissertation Universität München, 1999. Available via `http://www.tcs.informatik.uni-muenchen.de/~matthes/dissertation/matthesdiss.ps.gz`

[12] Ralph Matthes. Monotone (co)inductive types and positive fixed-point types. In *Theoretical Informatics and Applications* 33(4-5) pp. 309-328. EDP Sciences. 1999.

[13] Ralph Matthes. Monotone fixed-point types and strong normalization. In Georg Gottlob, Etienne Grandjean, and Katrin Seyr, editors, Computer Science Logic, 12th International Workshop, Brno, Czech Republic, August 24-28, 1998, Proceedings, volume 1584 of Lecture Notes in Computer Science, pages 298-312. Springer Verlag, 1999.

[14] Ralph Matthes. Monotone inductive and coinductive constructors of rank 2. In *Proceedings of Computer Science Logic 2001*. LNCS 2142. pp. 600-614. Springer Verlag. 2001.

[15] Ralph Matthes. Non-Strictly Positive Fixed-Points for Classical Natural Deduction. In *Annals of Pure and Applied Logic* 133. pp. 205-230. Elsevier 2005.

[16] N.P. Mendler. Recursive Types and Type Constraints in Second-Order Lambda Calculus. In *Proceedings of the 2nd Annual Symposium on Locig in Computer Science,Ithaca N.Y.* pp. 30-36 IEEE Computer Society Press, Washington D.C. 1987.

[17] N.P. Mendler. Inductive Types and Type Constraints in the Second-Order Lambda Calculus. *Annals of Pure and Applied Logic* **51**(1-2) pp. 159-172. North-Holland 1991.

[18] F. E. Miranda-Perea. On Extensions of AF2 with Monotone and Clausular (Co)inductive Definitions. Ph.D. Thesis, Ludwig-Maximilians-Universität München. Germany 2004.

[19] M. Parigot, Recursive programming with proofs. In *Theoretical Computer Science* **94**, pp.335-356. Elsevier. 1992.

[20] E. Poll, J. Zwanenburg. From Algebras and Coalgebras to Dialgebras. In *Coalgebraic Methods in Computer Science (CMCS'2001)*. Electronic Notes in Theoretical Computer Science **44**. Elsevier, 2001.

[21] J.J.M.M. Rutten. Automata and coinduction (an exercise in coalgebra). In Proceedings of CONCUR '98, D. Sangiorigi and R. de Simone (eds.), LNCS 1466, Springer, 1998, pp. 194-218.

[22] T. Uustalu, V. Vene. Primitive (co)recursion and course-of-value (co)iteration, categorically. In *INFORMATICA*, v. 10, n.1, pp. 5-26. 1999.

[23] T. Uustalu, V. Vene. Least and greatest fixed-points in intuitionistic natural deduction. In *Theoretical Computer Science*, v. 272, n. 1-2, pp. 315-339, 2002.

[24] V. Vene. Categorical programming with inductive and coinductive types. Diss. Math. Uni. Tartuensis, v. 23, Uni. of Tartu, Aug. 2000.

[25] G.C. Wraith. A note on categorical datatypes. In D.Pitts et al, editors. Category Theory and Computer Science. LNCS **389**, Springer Verlag 1989.

Appendix A. Syntactical Sugar

Future work includes the implementation of our type system MCICT, here we propose some syntactical sugar to avoid the heavy use of λ-terms.

- Data type definition.
  An inductive type $\mu X(F_1, \ldots, F_n)$ can be declared as:

$$\begin{array}{rcl} < type\_name > & = & \texttt{Inductive } X \texttt{ with constructors} \\ & & \quad c_1 : F_1 \to X \\ & & \quad \vdots \\ & & \quad c_n : F_n \to X \end{array}$$

  Here c_i are particular names for the constructors, $c_i := \mathsf{in}_{k,i}$
  A Coinductive type $\nu X(F_1, \ldots, F_n)$ is declared as:

$$\begin{array}{rcl} < type\_name > & = & \texttt{Coinductive } X \texttt{ with destructors} \\ & & \quad d_1 : X \to F_1 \\ & & \quad \vdots \\ & & \quad d_n : X \to F_n \end{array}$$

  Here d_i are particular names for the destructors, $d_i := \mathsf{out}_{k,i}$
- A function $\mathsf{fun} : \mathcal{I} \to B$, where $\mathcal{I} := \mu X(F_1, \ldots, F_k)$, defined by iteration $\mathsf{fun} := \lambda x.\mathsf{It}_k(\vec{m}, \vec{s}, x)$ is represented as:

$$\begin{array}{rcl} \mathsf{fun} & = & \texttt{iterator of } \mathcal{I} \texttt{ to } B \texttt{ with steps} \\ & & \quad s_1 : F_1[X := B] \to B \\ & & \quad \vdots \\ & & \quad s_k : F_k[X := B] \to B \\ & & \texttt{where } \mathsf{map}_1 = m_1, \ldots \mathsf{map}_k = m_k \end{array}$$

  Analogously if $\mathsf{fun} := \lambda x.\mathsf{Rec}_k(\vec{m}, \vec{s}, x)$ we just change the word "`iterator`"to "`recursor`"
- A function $\mathsf{fun} : B \to \mathcal{C}$, where $\mathcal{C} := \nu X(F_1, \ldots, F_k)$, defined by corecursion $\mathsf{fun} := \lambda x.\mathsf{CoRec}_k(\vec{m}, \vec{s}, x)$ is represented as:

$$\begin{array}{rcl} \mathsf{fun} & = & \texttt{corecursor of } \mathcal{C} \texttt{ from } B \texttt{ with steps} \\ & & \quad s_1 : B \to F_1[X := B] \\ & & \quad \vdots \\ & & \quad s_k : B \to F_k[X := B] \\ & & \texttt{where } \mathsf{map}_1 = m_1, \ldots \mathsf{map}_k = m_k \end{array}$$

  Analogously if $\mathsf{fun} := \lambda x.\mathsf{Colt}_k(\vec{m}, \vec{s}, x)$ we just change the word "`corecursor`"to "`coiterator`"
- The operational semantics is then, as follows:
  - If fun is defined by iteration then $\mathsf{fun}(c_i x) \to s_i\big(\mathsf{map}_i\, \mathsf{fun}\, x\big)$
  - If fun is defined by recursion then $\mathsf{fun}(c_i x) \to s_i\big(\mathsf{map}_i\, \langle \mathsf{Id}, \mathsf{fun}\rangle\, x\big)$
  - If fun is defined by corecursion then $d_i(\mathsf{fun}\, x) \to \mathsf{map}_i\, [\mathsf{Id}, \mathsf{fun}]\, (s_i x)$
  - If fun is defined by coiteration then $d_i(\mathsf{fun}\, x) \to \mathsf{map}_i\, \mathsf{fun}\, (s_i x)$

Appendix B. Canonical Monotonicity Witnesses

We present a canonical selection for monotonicity witnesses which essentially corresponds to the usual definitions for the positive cases, we do not restrict ourselves to strict positivity and define also antimonotonicity. Moreover we define witnesses for interleaving types.

Definition 28 (Antimonotonicity)**.** *Given a type A and a type variable X, we define the type $F\,\mathsf{mon}^-\,X$ as:*

$$F\,\mathsf{mon}^-\,X := \forall X.\forall Y.(X \to Y) \to F[X := Y] \to F$$

If a term m inhabits the type $F\,\mathsf{mon}^-\,X$ in a given context, then the functor $\langle \lambda X F, m\rangle$ will be antimonotone (contravariant) in the same context.

Definition 29 (Generic (Anti)monotonicity Witnesses)**.** *We define the following* MCICT*-terms:*

- $\mathbb{M}_{\mathsf{id}} := \lambda x x$
- $\mathbb{M}_{\mathsf{triv}} := \lambda f \lambda x x$
- $\mathbb{M}_{\rightarrow} := \lambda m_1 \lambda m_2 \lambda f \lambda x \lambda y. m_2 f (x (m_1 f y))$
- $\mathbb{M}_{\forall} := \lambda m \lambda f \lambda x. m f x$
- $\mathbb{M}_{\times} := \lambda m_1 \lambda m_2 \lambda f \lambda x \langle m_1 f (\mathsf{fst}\, x), m_2 f (\mathsf{snd}\, x)\rangle$
- $\mathbb{M}_{+} := \lambda m_1 \lambda m_2 \lambda f \lambda x. \mathsf{case}(x, y.\, \mathsf{inl}\, m_1 f y, z.\, \mathsf{inr}\, m_2 f z)$
- $\mathbb{M}^k_{\mu} := \lambda \vec{m} \lambda \vec{n} \lambda f \lambda x. \mathsf{It}_k(\vec{m}, \vec{s}, x)$, *where* $s_i := \lambda z.\, \mathsf{in}_{k,i}(n_i f z)$.
- $\mathbb{M}^k_{\nu} := \lambda \vec{m} \lambda \vec{n} \lambda f \lambda x.\, \mathsf{out}_k^{-1}(\vec{m}, \vec{s})$ *where* $s_i := n_i f (\mathsf{out}_{k,i}\, x)$.

Proposition 13 (Derived Typing Rules for (Anti)monotonicity)**.** *The following rules are derivable:*

- $\Gamma \vdash \mathbb{M}_{\mathsf{id}} : X\,\mathsf{mon}\,X$
- *If* $X \notin FV(F)$ *then* $\Gamma \vdash \mathbb{M}_{\mathsf{triv}} : F\,\mathsf{mon}\,X$ *and* $\Gamma \vdash \mathbb{M}_{\mathsf{triv}} : F\,\mathsf{mon}^-\,X$
- *If* $\Gamma \vdash m_1 : F\,\mathsf{mon}^-\,X$ *and* $\Gamma \vdash m_2 : G\,\mathsf{mon}\,X$ *then*

$$\Gamma \vdash \mathbb{M}_{\rightarrow} m_1 m_2 : (F \rightarrow G)\,\mathsf{mon}\,X$$

- *If* $\Gamma \vdash m_1 : F\,\mathsf{mon}\,X$ *and* $\Gamma \vdash m_2 : G\,\mathsf{mon}^-\,X$ *then*

$$\Gamma \vdash \mathbb{M}_{\rightarrow} m_1 m_2 : (F \rightarrow G)\,\mathsf{mon}^-\,X$$

- *If* $\Gamma \vdash t : \forall Z.F\,\mathsf{mon}\,X$ *then* $\Gamma \vdash \mathbb{M}_{\forall} t : (\forall Z F)\,\mathsf{mon}\,X$
- *If* $\Gamma \vdash t : \forall Z.F\,\mathsf{mon}^-\,X$ *then* $\Gamma \vdash \mathbb{M}_{\forall} t : (\forall Z.F)\,\mathsf{mon}^-\,X$
- *If* $\Gamma \vdash m_1 : F\,\mathsf{mon}\,X$ *and* $\Gamma \vdash m_2 : G\,\mathsf{mon}\,X$ *then*

$$\Gamma \vdash \mathbb{M}_{\times} m_1 m_2 : (F \times G)\,\mathsf{mon}\,X$$

- *If* $\Gamma \vdash m_1 : F\,\mathsf{mon}^-\,X$ *and* $\Gamma \vdash m_2 : G\,\mathsf{mon}^-\,X$ *then*

$$\Gamma \vdash \mathbb{M}_{\times} m_1 m_2 : (F \times G)\,\mathsf{mon}^-\,X$$

- *If* $\Gamma \vdash m_1 : F\,\mathsf{mon}\,X$ *and* $\Gamma \vdash m_2 : G\,\mathsf{mon}\,X$ *then*

$$\Gamma \vdash \mathbb{M}_{+} m_1 m_2 : (F + G)\,\mathsf{mon}\,X$$

- *If* $\Gamma \vdash m_1 : F\,\mathsf{mon}^-\,X$ *and* $\Gamma \vdash m_2 : G\,\mathsf{mon}^-\,X$ *then*

$$\Gamma \vdash \mathbb{M}_{+} m_1 m_2 : (F + G)\,\mathsf{mon}^-\,X$$

- *If* $\Gamma \vdash m_i : (\forall X.F_i\,\mathsf{mon}\,Z)$ *and* $\Gamma \vdash n_i : (\forall Z.F_i\,\mathsf{mon}\,X)$ *then*

$$\Gamma \vdash \mathbb{M}^k_{\mu} \vec{m} \vec{n} : \mu Z(F_1, \ldots, F_k)\,\mathsf{mon}\,X$$

- *If* $\Gamma \vdash m_i : (\forall X.F_i\,\mathsf{mon}\,Z)$ *and* $\Gamma \vdash n_i : (\forall Z.F_i\,\mathsf{mon}^-\,X)$ *then*

$$\Gamma \vdash \mathbb{M}^k_{\mu} \vec{m} \vec{n} : \mu Z(F_1, \ldots, F_k)\,\mathsf{mon}^-\,X$$

- *If* $\Gamma \vdash m_i : (\forall X.F_i\,\mathsf{mon}\,Z)$ *and* $\Gamma \vdash n_i : (\forall Z.F_i\,\mathsf{mon}\,X)$ *then*

$$\Gamma \vdash \mathbb{M}^k_{\nu} \vec{m} \vec{n} : \nu Z(F_1, \ldots, F_k)\,\mathsf{mon}\,X$$

- *If* $\Gamma \vdash m_i : (\forall X.F_i\,\mathsf{mon}\,Z)$ *and* $\Gamma \vdash n_i : (\forall Z.F_i\,\mathsf{mon}^-\,X)$ *then*

$$\Gamma \vdash \mathbb{M}^k_{\nu} \vec{m} \vec{n} : \nu Z(F_1, \ldots, F_k)\,\mathsf{mon}^-\,X$$

Communicated by (The editor will be set by the publisher).
August 13, 2008.