Pure Type Systems with Corecursion on Streams

From Finite to Infinitary Normalisation

Paula Severi
Department of Computer Science, University of Leicester, UK
ps56@mcs.le.ac.uk

Fer-Jan de Vries
Department of Computer Science, University of Leicester, UK
fdv1@mcs.le.ac.uk

Abstract

In this paper, we use types for ensuring that programs involving streams are well-behaved. We extend pure type systems with a type constructor for *streams*, a modal operator *next* and a fixed point operator for expressing *corecursion*. This extension is called *Pure Type Systems with Corecursion* (CoPTS). The typed lambda calculus for reactive programs defined by Krishnaswami and Benton can be obtained as a CoPTS. CoPTS's allow us to study a wide range of typed lambda calculi extended with corecursion using only one framework. In particular, we study this extension for the calculus of constructions which is the underlying formal language of Coq. We use the machinery of infinitary rewriting and formalize the idea of well-behaved programs using the concept of infinitary normalization. We study the properties of infinitary weak and strong normalization for CoPTS's. The set of finite and infinite terms is defined as a metric completion. We shed new light on the meaning of the modal operator by connecting the modality with the depth used to define the metric. This connection is the key to the proofs of infinitary weak and strong normalization.

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

General Terms term1, term2

Keywords Typed lambda calculus, recursion, streams, infinitary normalisation

1. Introduction

In this paper, we are interested in using types to ensure that programs involving streams defined by recursive equations are well-behaved. As an example, we consider streams in Haskell. The program zeros defined by the following corecursive equation:

```
zeros = 0:zeros
```

is well-behaved because the run-time system yields a value which is a potentially infinite normal form:

$$0 : (0 : (0 : (\ldots)))$$

The following programs are not well-behaved because they do not produce any output.

```
omega = omega
omegaprime = tail (0: omegaprime)
e = filter (\x-> (x>0)) zeros
```

The last one does not produce the empty list but it loops as the other two. Intuitively, the above programmes are "badly behaved". The idea of badly behaved programmes is formalized in infinitary rewriting through the concept of infinitary normalization [28, 29, 32]. We say that a programme is *infinitary normalizing* if it has either a finite or an infinite normal form. None of the above three examples are infinitary normalizing. A typed lambda calculus satisfies the property of *infinitary (weak) normalization* if all typable terms are infinitary normalizing. Unfortunately, the typed lambda calculus underlying Haskell is not infinitary normalizing since it allows us to type the above terms which are not infinitary normalizing.

```
omega :: a
omegaprime :: [Integer]
e :: [Integer]
```

The proof assistant Coq disallows badly behaved programs by using the so-called *guardedness condition* for corecusive definitions, i.e. the recursive calls should be guarded by constructors [10, 22]. For example, this condition allows us to type programs on streams such as:

```
interleave xs ys = (head xs):
                      (interleave ys (tail xs))
```

Programs defined using the guardedness condition are infinitary normalizing. However, the guardedness condition has its limitations. When constructing proofs in Coq, the guardedness condition can only be verified when the pretended proof has been completed [21]. It is also quite strong and forbids to type well-behaved programs like the following one.

```
zerosprime = 0: (interleave zerosprime zerosprime)
```

The typed lambda calculus of reactive programs defined by Krishnaswami and Benton can type zerosprime [36]. This system is the simply typed lambda calculus extended with corecursion on streams. In this paper, we extend the typed lambda calculus of reactive programs to the calculus of constructions which is a subset of the underlying formal language for Coq [12]. This extension will allow us to write other forms of abstractions:

1. Polymorphic functions such as map and zip.
2. Type constructors such as the following one (written in Haskell notation):

```
type DoubleFun a = [a] -> [a] -> [a]
```

3. Properties on streams and their proofs, using the Curry Howard isomorphism [13, 15, 25]. For example, we can have a constant
$$\mathsf{EqStr} : \Pi X{:}\mathsf{set}.(\mathsf{Stream}\ X) \to (\mathsf{Stream}\ X) \to \mathsf{prop}$$
to represent equality between streams.

To give a more general presentation, we consider pure type systems (PTS's) [2, 5, 49]. Pure type systems are a framework to define several existing typed lambda calculi à la Church in a uniform way [1]. In particular, this includes systems with dependent types and an infinite type hierarchy such as the extended calculus of construction (also a subset of the underlying formal language of Coq) [41]. We define *Pure Type Systems with Corecursion* (CoPTS's) by first extending the set of pseudoterms of a PTS with:

1. The type for *streams* $(\mathsf{Stream}\ A)$ with the constructor cons and the destructors head and tail.
2. The *next* modal type $(\bullet A)$, the constructor $\circ$ and a destructor await.
3. The fixed point operator to express *corecursion* which is denoted by cofix.

The judgements of a CoPTS are written as $\Gamma \vdash a{:}_i A$ where i is an index representing *time*. A term of type $(\bullet A)$ represents 'the information that is going to be displayed *later* in the future'.

CoPTS's allow us to study a wide range of typed lambda calculi extended with corecursion using only one framework. We will prove infinitary weak normalization for CoPTS's in a general way that does not depend on the type system. To be more precise, we give conditions on the PTS that ensure that the corresponding CoPTS is infinitary weakly normalizing. This general result applies to the extended calculus of constructions. We also study the property of infinitary strong normalization. Infinitary strong normalization is the analogon of strong normalization in the finitary setting. To prove these results, we use the machinery of infinitary rewriting [28, 29, 32].

What do the infinite normal forms of typable terms look like?

To describe the infinite normal forms of the typable terms, we define a set $\mathcal{C}^\infty$ of finite and infinite terms as a metric completion using an appropriate *metric*. This metric uses a notion of *depth* where the depth of b in argument positions in $(\mathsf{cons}\ a\ b)$ and $(\circ b)$ is counted one deeper than the depth of the terms $(\mathsf{cons}\ a\ b)$ and $(\circ b)$ themselves. As a result, the set $\mathcal{C}^\infty$ is strictly included in any of the sets of finite and infinite terms defined for infinitary term rewriting systems, infinitary lambda calculus and infinitary combinatory reduction systems [28, 29, 32].

To prove infinitary weak normalization, we need to find a strategy of reduction that finds an infinite normal form.

What is an infinitary normalizing strategy?

Unlike finite lambda calculus, the leftmost strategy is not infinitary normalizing. In the example,

```
zeross = zeros: zeross
```

the leftmost strategy does not lead to the infinite normal form in ω-steps [2]. We follow an infinitary normalizing strategy that reaches the normal form in ω-steps which is a variation of the *depth-first leftmost strategy*. Figure 1 shows a tree representation of the infinite normal form of zeross that respects our notion of depth. The tree is finitely branched. The first line is at depth 0 and it should be printed first, the second line is at depth 1 and it should be printed second, and so on.

[1] By typing à la Church, we mean that abstractions are of the form $\lambda x{:}A.b$, i.e. the variable in the abstraction is provided with an explicit type declaration.

[2] In the infinitary lambda calculus, the situation is actually worse: there are terms that do not have any leftmost redex at all. These terms are of the form $((\ldots)P_2)P_1$, called *infinite left spines*.

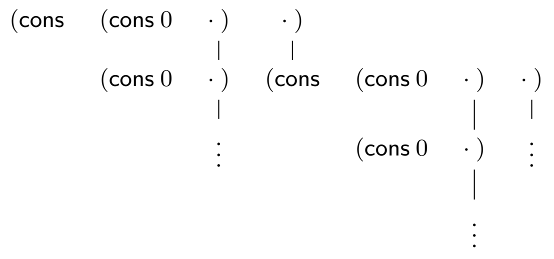

Figure 1. The infinite normal form of zeross represented as a tree

We shed new light on the meaning of the modal operator by connecting it with the depth used to define our metric. The connection between the modal operator and the depth is the key to the proofs of infinitary weak and strong normalization. A programming language will never be able to display the whole infinite normal form $0 : (0 : (0 : (\ldots)))$ but it will display only its truncation at certain depth n (an approximant):
$$0 : (0 : \ldots (0 : \bot) \ldots))$$
The modal operator $(\bullet A)$ represents the information that will appear *later* in the computation which is also the information that appears *deeper* in the infinite normal form.

The connection between the modality and the depth is formalized as follows.

If $x{:}_i(\bullet A) \vdash b{:}_i B$ then all occurrences of x in b occur at depth (strictly) greater than 0.

Similarly,

If $x{:}_{i+1} A \vdash b{:}_i B$ then all occurrences of x in b occur at depth (strictly) greater than 0.

For typing $(\mathsf{cofix}\ x{:}A.b)$, we require that $x{:}_{i+1}A \vdash b{:}_i A$. This means that the variable x in $(\mathsf{cofix}\ x{:}A.b)$ occurs in b at depth (strictly) greater than 0. In other words, the truncation of b at depth 1 contains no occurrences of x. Let's examine what happens during the computation. Let $\to_\gamma$ be the reduction that unfolds fixed points:
$$(\mathsf{cofix}\ x{:}A.b) \to_\gamma b[x := (\mathsf{cofix}\ x{:}A.b)]$$
After contracting the fixed point, we have that the truncation of $b[x := (\mathsf{cofix}\ x{:}A.b)]$ at depth 1 does not contain any residuals of the contracted redex. As an example, we consider the programme zeros which is expressed in our syntax as follows.
$$\mathsf{zeros} = (\mathsf{cofix}\ xs{:}(\mathsf{Stream\ Nat}).xs)$$
The γ-redex occurs at depth 0 in zeros. We perform one γ-reduction step:
$$\mathsf{zeros} \quad \to_\gamma \quad (\mathsf{cons}\ 0\ \mathsf{zeros})$$
In the *future* (after contracting the γ-redex), the γ-redex occurs at depth 1. The truncation of $(\mathsf{cons}\ 0\ \mathsf{zeros})$ at depth 1 which is $(\mathsf{cons}\ 0\ \bot)$ represents the information that has been displayed so far. The truncated subterm zeros which is at depth 1 in $(\mathsf{cons}\ 0\ \mathsf{zeros})$ represents the information that will appear *later* which also appears *deeper*.

This paper is organized as follows. Section 2 gives an overview of PTS's. Section 3 defines the notion of CoPTS's. Section 4 shows some basic properties, the most important one concerns $\beta\sigma$-strong normalization. Section 5 defines the set $\mathcal{C}^\infty$ of finite and infinite terms as metric completion of the set of finite terms. Section 6 studies infinitary weak normalization. Section 7 studies infinitary strong normalization. Section 8 draws some conclusions and explains related work. Section 9 gives some plan for future work.

2. Preliminaries on Pure Type Systems

In this section, we recall the notion of *pure type system* (PTS) [2]. They were introduced independently by Berardi and Terlouw [5, 49] as a way of generalizing the systems of the λ-cube [2]. Pure type systems consists of only seven typing rules parametrized by a certain *specification*. There are only two rules which are parametric: the axiom and the product rule. By instantiating the parameters, we can describe a large class of typed lambda calculi such as the the extended calculus of constructions [41] and even inconsistent systems [23]. The word *pure* stands for the fact that there is only one type constructor and only one reduction, namely Π and β.

We recall the definition of specification. The specification fixes the parameters in the definition of pure type system.

Definition 2.1 (Specification)**.** *A* specification *is a triple $\mathcal{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ such that*

1. *$\mathcal{S}$ is a set of symbols called* sorts,
2. *$\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ called set of* axioms,
3. *$\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ called set of* rules.

We will need the notion of single sorted specification to ensure unicity of types (Theorem 4.6) and the well-definedness of the encoding in $\lambda\omega$ (Definition 4.9).

Definition 2.2 (Single Sorted Specification)**.** *We say that a specification is single sorted if*

1. *If (s_1, s_2) and (s_1, s_2') are in $\mathcal{A}$ then $s_2 = s_2'$.*
2. *If (s_1, s_2, s_3) and (s_1, s_2, s_3') are in $\mathcal{R}$ then $s_3 = s_3'$.*

Types and terms are defined in the same set $\mathcal{T}$.

Definition 2.3 (Pseudoterms)**.** *The set $\mathcal{T}_\mathcal{S}$ (or $\mathcal{T}$ for short) of pseudoterms is defined as follows.*

$$\mathcal{T} ::= \mathcal{V} \mid \mathcal{S} \mid (\lambda\mathcal{V}{:}\mathcal{T}.\mathcal{T}) \mid (\mathcal{T}\ \mathcal{T}) \mid (\Pi\mathcal{V}{:}\mathcal{T}.\mathcal{T})$$

Sorts are denoted by $s, s', \ldots$, variables by $x, y, \ldots$ and pseudoterms by capital $A, B, \ldots$ and also by lower case $a, b, \ldots$. The set $\mathsf{fv}(A)$ of free variables of A is defined in the usual way and $A \to B$ is an abbreviation for $\Pi\ x{:}A.B$ if $x \notin \mathsf{fv}(B)$.

Definition 2.4 (β-Reduction)**.** *We define β-reduction as usual:*

$$(\lambda x{:}A.b)\ a \quad \to b[x := a] \quad (\beta)$$

The relation $\to_\beta$ is defined as the smallest relations on pseudoterms that are closed under the β-rule and under contexts.

In the following section, we will define other notions of reductions such as σ and γ. We introduce the following notation which works for all of them.

Notation 2.5. *Let ρ be a notion of reduction.*

1. *$M \to_\rho N$ denotes a one step reduction from M to N;*
2. *$M \twoheadrightarrow_\rho N$ denotes a finite reduction from M to N;*
3. *$M =_\rho N$ denotes conversion.*

A *pseudocontext* is a finite ordered sequence of type declarations: $\Gamma = x_1{:}A_1, x_2{:}A_2, \ldots x_n{:}A_n$ where x_i are all different variables and A_i are pseudoterms for all $1 \leq i \leq n$.

(axiom) $\vdash s_1{:}s_2 \quad \text{if } (s_1, s_2) \in \mathcal{A}$

(start) $\dfrac{\Gamma \vdash A{:}s}{\Gamma, x{:}A \vdash x{:}A}\ x\ \Gamma\text{-fresh}$

(weak) $\dfrac{\Gamma \vdash A{:}s \quad \Gamma \vdash b{:}B}{\Gamma, x{:}A \vdash b{:}B}\ x\ \Gamma\text{-fresh}$

(prod) $\dfrac{\Gamma \vdash A{:}s_1 \quad \Gamma, x{:}A \vdash B{:}s_2}{\Gamma \vdash (\Pi x{:}A.B){:}s_3}\ (s_1, s_2, s_3) \in \mathcal{R}$

(abs) $\dfrac{\Gamma, x{:}A \vdash b{:}B \quad \Gamma \vdash (\Pi x{:}A.B){:}s}{\Gamma \vdash (\lambda x{:}A.b){:}(\Pi x{:}A.B)}$

(app) $\dfrac{\Gamma \vdash b{:}(\Pi x{:}A.B) \quad \Gamma \vdash a{:}A}{\Gamma \vdash (b\ a){:}B[x := a]}$

(β-conv) $\dfrac{\Gamma \vdash a{:}A \quad \Gamma \vdash A'{:}s}{\Gamma \vdash a{:}A'}\ A =_\beta A'$

Figure 2. Pure Type Systems

Definition 2.6 (Pure Type System)**.** *A* Pure Type System *(PTS) denoted by $\lambda(\mathcal{S})$ is given by the judgement $\Gamma \vdash_\mathcal{S} a : A$ (or just $\Gamma \vdash a : A$) and defined by the typing rules of Figure 2.*

Notation 2.7. *The rule (s_1, s_2) is an abbreviation for (s_1, s_2, s_2).*

Example 2.8 (Systems of the λ-cube)**.** *The systems of the λ-cube are obtained from the following set of sorts and axioms [2].*

$$\mathcal{S} = \{\mathsf{type}, \mathsf{kind}\} \quad \mathcal{A} = \{(\mathsf{type}, \mathsf{kind})\}$$

The possibilities for $(s_1, s_2) \in \mathcal{R}$ for $s_1, s_2 \in \{\mathsf{type}, \mathsf{kind}\}$ are the following ones and each one allows us to represent different type of functions:

$(\mathsf{type}, \mathsf{type})$	*for terms depending on terms (functions),*
$(\mathsf{kind}, \mathsf{type})$	*for terms depending on types (polymorphic functions),*
$(\mathsf{type}, \mathsf{kind})$	*for types depending on terms (dependent types),*
$(\mathsf{kind}, \mathsf{kind})$	*for types depending on types (type constructors).*

The systems of the λ-cube consist of eight type systems. They all contain the rule $(\mathsf{type}, \mathsf{type})$*. The smallest set gives rise to the simply typed lambda calculus and the biggest one to the calculus of constructions, [12]. We show the specification of only four of these systems which will be used later.*

The simply typed lambda calculus $\lambda_\to$ is obtained from the specification $\mathcal{S}_\to$ defined by the common sets $\mathcal{S}$ and $\mathcal{A}$ given above for the systems of the λ-cube and the following set of rules:

$$\mathcal{R} = \{(\mathsf{type}, \mathsf{type})\}$$

The second order lambda calculus [23, 46] is the pure type system $\lambda 2$ obtained from the following set of rules:

$$\mathcal{R} = \{(\mathsf{type}, \mathsf{type}), (\mathsf{kind}, \mathsf{type})\}$$

The pure type system $\lambda\omega$ corresponds to $F\omega$ of [23] and is obtained from the following set of rules:

$$\mathcal{R} = \{(\mathsf{type}, \mathsf{type}), (\mathsf{kind}, \mathsf{type}), (\mathsf{kind}, \mathsf{kind})\}$$

The calculus of constructions [12] is obtained from the specification C which consists of the sets $\mathcal{S}$, $\mathcal{A}$ defined above and the following set of rules:

$$\mathcal{R} = \{(\mathsf{type}, \mathsf{type}), (\mathsf{kind}, \mathsf{type}), (\mathsf{type}, \mathsf{kind}), (\mathsf{kind}, \mathsf{kind})\}$$

Example 2.9 (Extended Calculus of Constructions as a PTS)**.** *The extended calculus of constructions [41] is obtained from the speci-*

fication EC defined as follows:

$$\begin{aligned}\mathcal{S} = &\ \{\mathsf{type}_n \mid n \in \mathbb{N}\}\\ \mathcal{A} = &\ \{(\mathsf{type}_n, \mathsf{type}_{n+1}) \mid n \in \mathbb{N}\}\\ \mathcal{R} = &\ \{(\mathsf{type}_n, \mathsf{type}_0, \mathsf{type}_0) \mid n \in \mathbb{N}\} \cup \\ &\ \{(\mathsf{type}_n, \mathsf{type}_m, \mathsf{type}_k) \mid m > 0 \& max(n, m) \leq k\}\}\end{aligned}$$

We see that $\lambda(EC)$ contains $\lambda(C)$ by identifying type_0 with type and type_1 with kind.

Example 2.10 (Inconsistent Pure Type Systems)**.** *The system λV is given by the following specification (called $\lambda*$ in [2]).*

$$\mathcal{S} = \{\mathsf{type}\} \quad \mathcal{A} = \{(\mathsf{type}, \mathsf{type})\} \quad \mathcal{R} = \{(\mathsf{type}, \mathsf{type})\}$$

This system is inconsistent in the sense that all types are inhabited [2, 23]. For examples where the circularity type:type *is not necessary to derive inconsistency, see [2, Example 5.2.4]. In any inconsistent logical pure type system, a looping combinator can be derived from any term of type $\bot = \Pi x{:}\mathsf{type}.x$ [11]. The paper [19] shows that Curry's and Turing's fixed point combinators $\mathsf{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ and $\Theta = (\lambda x f.f(xxf))(\lambda x f.f(xxf)$ cannot be typed in λV.*

Definition 2.11 (Term and Context)**.** *Let $\mathcal{S}$ be a specification.*

1. *A (typable) term is a pseudoterm a such that $\Gamma \vdash a{:}A$ for some Γ and A.*
2. *A (legal) context is a pseudocontext Γ such that $\Gamma \vdash a{:}A$ for some a and A.*

In the following definition, we consider an arbitrary reduction ρ. In later sections, we will define other notions of reductions besides β.

Definition 2.12 (Weak and Strong Normalization)**.** *Let ρ be a notion of reduction.*

1. *We say that a pseudoterm a is weakly ρ-normalizing if there exists a pseudoterm b in ρ-normal form such that $a \twoheadrightarrow_\rho b$.*
2. *We say that a pseudoterm a is strongly ρ-normalizing if all ρ-reduction sequences starting from a are finite.*

Definition 2.13 (Weakly and Strongly Normalizing PTS)**.** *We say that $\lambda(\mathcal{S})$ is strongly (weakly) β-normalizing if for all $\Gamma \vdash a{:}A$ we have that a and A are strongly (weakly) β-normalizing.*

Notation 2.14. *1. $\lambda(\mathcal{S}) \models \rho$-WN if $\lambda(\mathcal{S})$ is weakly ρ-normalizing.*
2. $\lambda(\mathcal{S}) \models \rho$-SN if $\lambda(\mathcal{S})$ is strongly ρ-normalizing.

Obviously, $\lambda(\mathcal{S}) \models \rho$-SN implies $\lambda(\mathcal{S}) \models \rho$-WN. The following result is proved in [41].

Theorem 2.15 (Strong Normalization of $\lambda(EC)$)**.** *We have that $\lambda(EC) \models \beta$-SN.*

The following result is proved in [2, Proposition 5.2.31]. We use the abbreviation $\bot = \Pi x{:}\mathsf{type}.x$.

Theorem 2.16 (Inconsistent implies not normalizing)**.** *Let $\lambda(\mathcal{S})$ be a PTS extending $\lambda 2$. Suppose $\Gamma \vdash a{:}\bot$. Then, a is not weakly β-normalizing. Hence, $\lambda(\mathcal{S}) \not\models \beta$-WN.*

As a consequence of the previous theorem, the inconsistent pure type system λV from Example 2.10 is not weakly normalizing.

3. Pure Type Systems with Corecursion

In this section, we define the notion of *pure type system with corecursion* (CoPTS). The set $\mathcal{T}$ of pseudoterms is extended to include the type constructor $(\mathsf{Stream}\ A)$ for *streams* of type A, the modal type $\bullet A$ *next* and a fixed point operator $(\mathsf{cofix}\ x{:}A.a)$ for expressing *corecursion*.

Definition 3.1 (Pseudoterms with Streams and Corecursion)**.** *The set $\mathcal{C}_\mathcal{S}$ (or $\mathcal{C}$ for short) is defined by the following grammar.*

$$\begin{aligned}\mathcal{C} ::= &\ \mathcal{V} \mid \mathcal{S} \mid (\lambda \mathcal{V}{:}\mathcal{C}.\mathcal{C}) \mid (\mathcal{C}\ \mathcal{C}) \mid (\Pi \mathcal{V}{:}\mathcal{C}.\mathcal{C})\\ &\ \bullet\mathcal{C} \mid \circ\mathcal{C} \mid (\mathsf{await}\ \mathcal{C}) \mid \\ &\ (\mathsf{Stream}\ \mathcal{C}) \mid (\mathsf{cons}\ \mathcal{C}\ \mathcal{C}) \mid (\mathsf{hd}\ \mathcal{C}) \mid (\mathsf{tl}\ \mathcal{C}) \mid \\ &\ (\mathsf{cofix}\ \mathcal{V}{:}\mathcal{C}.\mathcal{C})\end{aligned}$$

We introduce two notions of reductions apart from β: σ for computing the head and tail of a stream and γ for unfolding fixed points.

Definition 3.2 (σ and γ-Reductions)**.** *We define the following reduction rules:*

$$\begin{array}{lll}(\mathsf{await}\ (\circ a)) & \to a & (\sigma)\\ (\mathsf{hd}\ (\mathsf{cons}\ a\ b)) & \to a & (\sigma)\\ (\mathsf{tl}\ (\mathsf{cons}\ a\ b)) & \to b & (\sigma)\\ (\mathsf{cofix}\ x{:}A.b) & \to b[x := (\mathsf{cofix}\ x{:}A.b)] & (\gamma)\end{array}$$

The relations $\to_\sigma$, $\to_\gamma$ are defined as the smallest relations on pseudoterms that are closed under the respective rules and under contexts. The relation $\to_{\beta\sigma\gamma}$ is the union of $\to_\beta$, $\to_\sigma$ and $\to_\gamma$.

Judgements of CoPTS's are of the form $\Gamma \vdash a :_i A$ where i is an index representing "time". A pseudocontext

$$\Gamma = x_1{:}_{i_1}A_1, x_2{:}_{i_2}A_2, \ldots x_n{:}_{i_n}A_n$$

for a CoPTS is a finite ordered sequence of type declarations where x_i are all different variables and A_i are pseudoterms in $\mathcal{C}$ for all $1 \leq i \leq n$.

We extend the typing rules of pure type systems for our extended set $\mathcal{C}$ of pseudoterms. Recall that $\mathcal{S}_\to$ is the specification for the simply typed lambda calculus defined in Example 2.8.

Definition 3.3 (Pure Type System with Corecursion)**.** *Let $\mathcal{S}$ be a specification extending $\mathcal{S}_\to$. A* Pure Type System with Corecursion on Streams *(CoPTS) denoted by $\lambda^{\mathsf{co}}(\mathcal{S})$ is given by the judgement $\Gamma \vdash^{\mathsf{co}}_{\mathcal{S}} a :_i A$ (or just $\Gamma \vdash a{:}_i A$) for $i \in \mathbb{N}$ and defined by the typing rules of Figure 3.*

Example 3.4 (Typed λ-calculus of Reactive Programs as a CoPTS)**.** *Krishnaswami and Benton's typed lambda calculus presented in [36] can be obtained as a CoPTS using the specification of the simply typed lambda calculus given in Example 2.8. This system will be denoted as $\lambda^{\mathsf{co}}_{\to}$.*

Remark 3.5 (Alternative Typing Rules for cofix using Modality)**.** *As in [36, 37], we add a constant* cofix *that represents the fixed point combinator. Our typing rule for (**cofix**) in Figure 3 is similar to the one presented in [37]. In this version of the rule, the variable x needs to have type A using the index $i + 1$. There is another version of the rule that uses modality $\bullet A$ and it is as follows.*

$$(\textbf{cofix'})\ \frac{\Gamma, x{:}_i\bullet A \vdash b{:}_i A \quad \Gamma \vdash A{:}_i\mathsf{type}}{\Gamma \vdash \mathsf{cofix}'\ x{:}\bullet A.b{:}_i A}$$

*The typing rules (*cofix*) and (*cofix*') are equivalent. The rule (*cofix*) allows us to derive (*cofix'*) by defining $\mathsf{cofix}'\ x{:}\bullet A.b = \mathsf{cofix}\ y{:}A.b[x := (\circ y)]$. Conversely, we can set $\mathsf{cofix}\ y{:}A.b = \mathsf{cofix}'\ x{:}\bullet A.b[y := (\mathsf{await}\ x)]$ and hence both systems are equivalent. It is also easy to see that the typing rule for* cofix*' is equivalent to adding a type declaration of the form $\mathsf{cofix}'' :_i (\bullet A \to A) \to A$ for all i as in [36].*

In spite of the fact that the rules (**cofix**) and (**cofix**') are equivalent, we prefer the rule (**cofix**) to (**cofix**'). The terms that will be shown later in our examples are typed using (**cofix**) and we see that in these examples the modality is not necessary. If we had defined the type system using the rule (**cofix**'), our programmes would have

$$\textbf{(axiom)}\ \vdash s_1{:_i}s_2 \quad \text{if } (s_1, s_2) \in \mathcal{A}$$

$$\textbf{(start)}\ \frac{\Gamma \vdash A{:_i}\mathsf{s} \quad j \geq i}{\Gamma, x{:_i}A \vdash x{:_j}A}\ x\ \Gamma\text{-fresh}$$

$$\textbf{(weak)}\ \frac{\Gamma \vdash A{:_i}\mathsf{s} \quad \Gamma \vdash b{:_j}B}{\Gamma, x{:_i}A \vdash b{:_j}B}\ x\ \Gamma\text{-fresh}$$

$$\textbf{(prod)}\ \frac{\Gamma \vdash A{:_i}s_1 \quad \Gamma, x{:_i}A \vdash B{:_i}s_2}{\Gamma \vdash (\Pi x{:}A.B){:_i}s_3}\ (s_1, s_2, s_3) \in \mathcal{R}$$

$$\textbf{(abs)}\ \frac{\Gamma, x{:_i}A \vdash b{:_i}B \quad \Gamma \vdash (\Pi x{:}A.B){:_i}\mathsf{s}}{\Gamma \vdash (\lambda x{:}A.b){:_i}(\Pi x{:}A.B)}$$

$$\textbf{(app)}\ \frac{\Gamma \vdash b{:_i}(\Pi x{:}A.B) \quad \Gamma \vdash a{:_i}A}{\Gamma \vdash (b\ a){:_i}B[x := a]}$$

$$(\beta\sigma\gamma\textbf{-conv})\ \frac{\Gamma \vdash a{:_i}A \quad \Gamma \vdash A'{:_i}s}{\Gamma \vdash a{:_i}A'}\ \ A{=_{\beta\sigma\gamma}}A'$$

$$\textbf{(mod)}\ \frac{\Gamma \vdash A{:_i}\mathsf{type}}{\Gamma \vdash \bullet A{:_i}\mathsf{type}}$$

$$(\bullet I)\ \frac{\Gamma \vdash a{:_{i+1}}A}{\Gamma \vdash \circ a{:_i}\bullet A}$$

$$(\bullet E)\ \frac{\Gamma \vdash a{:_i}\bullet A}{\Gamma \vdash (\mathsf{await}\ a){:_{i+1}}A}$$

$$\textbf{(stream)}\ \frac{\Gamma \vdash A{:_i}\mathsf{type}}{\Gamma \vdash (\mathsf{Stream}\ A){:_i}\mathsf{type}}$$

$$\textbf{(cons)}\ \frac{\Gamma \vdash a{:_i}A \quad \Gamma \vdash b{:_{i+1}}(\mathsf{Stream}\ A)}{\Gamma \vdash (\mathsf{cons}\ a\ b){:_i}(\mathsf{Stream}\ A)}$$

$$\textbf{(hd)}\ \frac{\Gamma \vdash a{:_i}(\mathsf{Stream}\ A)}{\Gamma \vdash (\mathsf{hd}\ a){:_i}A}$$

$$\textbf{(tl)}\ \frac{\Gamma \vdash a{:_i}(\mathsf{Stream}\ A)}{\Gamma \vdash (\mathsf{tl}\ a){:_{i+1}}(\mathsf{Stream}\ A)}$$

$$\textbf{(cofix)}\ \frac{\Gamma, x{:_{i+1}}A \vdash b{:_i}A \quad \Gamma \vdash A{:_i}\mathsf{type}}{\Gamma \vdash (\mathsf{cofix}\ x{:}A.b){:_i}A}$$

Figure 3. Pure Type Systems with Corecursion on Streams

been burdened with modalities. For example, let's write the example of zeros given in the introduction using cofix'.

$$\mathsf{zeros}'' = \quad (\mathsf{cofix}'\ xs{:}\bullet(\mathsf{Stream\ Nat}).(\mathsf{cons}\ 0\ (\mathsf{await}\ xs)))$$

The explicit type given for xs contains $\bullet$ and the recursive call needs to use await. None of this is necessary when zeros is written using cofix (see Example 3.6). This means that depending on the applications we may be able to remove the rules for modalities from our system. We include the modality to encompass the type system of reactive programs as a CoPTS [36] (examples where modalities are necessary can be found in [36–38]). Nakano's type system has modalities without indices but it makes use of subtyping and recursive types [44]. In our current formulation, the indices cannot be removed. But this does not matter, because the indices are hidden to the programmer as they are handled by the type checker.

We will formalize the Haskell programmes given in the introduction in our setting and show that the well-behaved ones are typable and the badly behaved ones are not. We define a context Γ_{Nat} containing the following type declarations:

$$\begin{array}{rl}
\mathsf{Nat} & :_i \mathsf{type} \\
0 & :_i \mathsf{Nat} \\
\mathsf{suc} & :_i \mathsf{Nat} \to \mathsf{Nat} \\
+ & :_i \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat} \\
* & :_i \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat} \\
\mathsf{Bool} & :_i \mathsf{type} \\
< & :_i \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Bool} \\
\mathsf{if} & :_i \mathsf{Bool} \to (\mathsf{Stream\ Nat}) \to (\mathsf{Stream\ Nat})
\end{array}$$

For the sake of the example, adding those constants to the context suffices. However, for a real programming language, we should add these constants to the syntax with the respective reduction and typing rules.

Example 3.6 (Terms typable in $\lambda^{\mathsf{co}}_{\to}$)**.** *Define the following:*

$$\begin{array}{rl}
\mathsf{FunSNat} = & (\mathsf{Stream\ Nat}) \to (\mathsf{Stream\ Nat}) \to (\mathsf{Stream\ Nat}) \\
\mathsf{zeros} = & (\mathsf{cofix}\ xs{:}(\mathsf{Stream\ Nat}).(\mathsf{cons}\ 0\ xs)) \\
\mathsf{interleave} = & \mathsf{cofix}\ f{:}\ \mathsf{FunSNat}. \\
& \quad \lambda xs : (\mathsf{Stream\ Nat}). \\
& \quad\quad \lambda ys : (\mathsf{Stream\ Nat}). \\
& \quad\quad\quad (\mathsf{cons}\ (\mathsf{hd}\ xs)\ (f\ ys\ (\mathsf{tl}\ xs))) \\
\mathsf{sumlist} = & \mathsf{cofix}\ f{:}\mathsf{FunSNat}. \\
& \quad \lambda xs{:}(\mathsf{Stream\ Nat}). \\
& \quad\quad \lambda ys{:}(\mathsf{Stream\ Nat}). \\
& \quad\quad\quad \mathsf{cons}\ (+\ (\mathsf{hd}\ xs)\ (\mathsf{hd}\ ys)) \\
& \quad\quad\quad\quad\quad (f\ (\mathsf{tl}\ xs)\ (\mathsf{tl}\ ys)) \\
\mathsf{merge} = & \mathsf{cofix}\ f{:}\mathsf{FunSNat}. \\
& \quad \lambda xs{:}(\mathsf{Stream\ Nat}). \\
& \quad\quad \lambda ys{:}(\mathsf{Stream\ Nat}). \\
& \quad\quad\quad \mathsf{if}\ (\mathsf{hd}\ xs) < (\mathsf{hd}\ ys)\ \mathsf{then} \\
& \quad\quad\quad\quad (\mathsf{cons}\ (\mathsf{hd}\ xs)\ (f\ (\mathsf{tl}\ xs)\ ys)) \\
& \quad\quad\quad \mathsf{elseif}\ (\mathsf{hd}\ xs) < (\mathsf{hd}\ ys)\ \mathsf{then} \\
& \quad\quad\quad\quad (\mathsf{cons}\ (\mathsf{hd}\ ys)\ (f\ xs\ (\mathsf{tl}\ ys))) \\
& \quad\quad\quad \mathsf{else} \\
& \quad\quad\quad\quad (\mathsf{cons}\ (\mathsf{hd}\ xs)\ (f\ (\mathsf{tl}\ xs)\ (\mathsf{tl}\ ys)))
\end{array}$$

We have that all the above terms can be typed in $\lambda^{\mathsf{co}}_{\to}$.

$$\begin{array}{ll}
\Gamma_{\mathsf{Nat}} & \vdash \mathsf{zeros} :_i (\mathsf{Stream\ Nat}) \\
\Gamma_{\mathsf{Nat}} & \vdash \mathsf{interleave} :_i \mathsf{FunSNat} \\
\Gamma_{\mathsf{Nat}} & \vdash \mathsf{sumlist} :_i \mathsf{FunSNat} \\
\Gamma_{\mathsf{Nat}} & \vdash \mathsf{merge} :_i \mathsf{FunSNat}
\end{array}$$

Example 3.7 (CoPTS's beyond $\lambda^{\mathsf{co}}_{\to}$)**.** *Going beyond $\lambda^{\mathsf{co}}_{\to}$ we can type polymorphic functions, type constructors and prove properties on streams using the Curry-Howard isomorphism. The polymorphic map function:*

$$\begin{array}{rl}
\mathsf{map} = & \lambda X{:}\mathsf{type}. \\
& \quad \lambda Y{:}\mathsf{type}. \\
& \quad\quad \lambda g{:}X \to Y. \\
& \quad\quad\quad \mathsf{cofix}\ f{:}(\mathsf{Stream}\ X) \to (\mathsf{Stream}\ Y). \\
& \quad\quad\quad\quad \lambda xs : (\mathsf{Stream}\ X). \\
& \quad\quad\quad\quad\quad (\mathsf{cons}\ (g\ (\mathsf{hd}\ xs))\ (f\ (\mathsf{tl}\ xs)))
\end{array}$$

can be typed in $\lambda^{\mathsf{co}}2$, i.e.

$$\begin{array}{l}
\vdash \mathsf{map}{:_i}\Pi X{:}\mathsf{type}.\Pi Y{:}\mathsf{type}. \\
\quad\quad (X \to Y) \to \\
\quad\quad (\mathsf{Stream}\ X) \to (\mathsf{Stream}\ Y)
\end{array}$$

We can also write type constructors such as:

$$\begin{array}{rl}
\mathsf{DoubleFun} = & \lambda X{:}\mathsf{type}. \\
& \quad (\mathsf{Stream}\ X) \to (\mathsf{Stream}\ X) \to (\mathsf{Stream}\ X)
\end{array}$$

which can be typed in $\lambda^{\mathsf{co}}\omega$ as follows.

$$\vdash \mathsf{DoubleFun} :_i \mathsf{type} \to \mathsf{type}$$

In $\lambda^{\mathsf{co}}(C)$, we can write and prove properties on streams. For example, we can have a constant EqStr *to represent equality between streams.*

$$\Gamma_{\mathsf{Nat}}, \mathsf{EqStr}{:_i}\Pi X{:}\mathsf{type}.(\mathsf{Stream}\ X) \to (\mathsf{Stream}\ X) \to \mathsf{type}$$
$$\vdash \mathsf{EqStr\ Nat\ zeros\ zeros'}{:_i}\mathsf{type}$$

Example 3.8 (CoPTS's type more than Coq)**.** *The proof assistant Coq ensures that corecursive definitions are well-defined by means of the the guardedness condition, i.e. the recursive calls should be guarded by constructors [10, 22]. The following programmes are not accepted by Coq. Let* mapn = map Nat Nat.

```
     zeros' =   (cofix $xs$:(Stream Nat).)
                  (cons 0 (interleave $xs$ $xs$)

        fib =   cofix $xs$:(Stream Nat).
                  (cons 1 (cons 1 (sumlist $xs$ (tl $xs$))))

    hamming =   cofix $h$:(Stream Nat).
                 cons 1
                     (merge
                       (mapn ($\lambda x$:Nat.$2 * x$) $h$)
                       (merge
                         (mapn($\lambda x$:Nat.$3 * x$) $h$)
                         (mapn ($\lambda x$:Nat.$5 * x$) $h$)))
```

They can all be typed in $\lambda^{\mathsf{co}}2$ as follows.

$$\begin{array}{ll} \Gamma_{\mathsf{Nat}} & \vdash \mathsf{zeros'} :_i \mathsf{Nat} \\ \Gamma_{\mathsf{Nat}} & \vdash \mathsf{fib} :_i \mathsf{Nat} \\ \Gamma_{\mathsf{Nat}} & \vdash \mathsf{hamming} :_i \mathsf{Nat} \end{array}$$

Example 3.9 (The Undesirables)**.** *The badly behaved programmes shown in the introduction can be written in our syntax as follows.*

$$\begin{array}{ll} \Omega & = (\mathsf{cofix}\ x{:}A.x) \\ \Omega_{\mathsf{tail}} & = (\mathsf{cofix}\ xs{:}A.(\mathsf{tl}\ xs)) \\ \Omega' & = (\mathsf{cofix}\ xs{:}A.(\mathsf{tl}\ (\mathsf{cons}\ 0\ xs))) \\ \Omega'' & = (\mathsf{cofix}\ x{:}A.(\mathsf{await}\ (\circ x))) \\ \mathsf{E} & = \mathsf{filter\ Nat}\ (\lambda xs{:}(\mathsf{Stream\ Nat}).x > 0)\ \mathsf{zeros}) \end{array}$$

where the function filter *is defined as follows:*

```
   filter =   $\lambda X$ : type.$\lambda P$ : $X \to$ Bool.
                cofix $f$:(Stream $X$) $\to$ (Stream $X$).
                  $\lambda xs$:(Stream $X$).
                    if ($P$ (hd $xs$)) then
                      (cons (hd $xs$) ($f$ (tl $xs$)))
                    else
                      ($f$ (tl $xs$))
```

None of the above terms are typable in any CoPTS. More formally, we have that the following holds for all A and i:

$$\begin{array}{ll} A :_i \mathsf{type} \not\vdash \Omega :_i A & A :_i \mathsf{type} \not\vdash \Omega_{\mathsf{tail}} :_i A \\ A :_i \mathsf{type} \not\vdash \Omega' :_i A & A :_i \mathsf{type} \not\vdash \Omega' :_i A \\ \Gamma_{\mathsf{Nat}} \not\vdash \mathsf{filter} ::_i A & \Gamma_{\mathsf{Nat}} \not\vdash \mathsf{E} ::_i A \end{array}$$

The terms $\Omega, \Omega_{\mathsf{tail}}$ and filter *are not typable because the depth of the variable for the fixed point operator happens to be at depth 0 (Theorem 7.6). The terms Ω' and Ω'' are not typable because they σ-reduce to Ω which is not typable (Theorem 4.5). The term* E *is not typable because it has a subterm which is not typable.*

Remark 3.10 (Why is the fixed point called cofix and not fix?)**.** *A basic primitive recursive function such as + defined as follows:*

```
         + =   ($\lambda x$ : Nat.
                 cofix$f$:(Nat $\to$ Nat).
                  $\lambda y$ : Nat.
                   case $y$ is 0          then $x$
                          is (succ $z$) then succ ($f$ $z$)
```

It is not typable because the variable f occurs at depth 0 (see Theorem 7.6). We think that the solution to this problem is to have two different fixed points, one for expressing recursion on inductive data types and the other one for corecursion on coinductive data types as in [20–22].

We define auxiliary type systems that will be used later in the proof of infinitary normalization.

Definition 3.11 (Pure Type System with Corecursion up to n)**.** *Let $\mathcal{S}$ be a specification extending $\mathcal{S}_{\to}$ and $n \in \mathbb{N}$. A* Pure Type System with Corecursion on Streams up to n *(CoPTSn) denoted by $\lambda^{\mathsf{co}}_{\mathsf{n}}(\mathcal{S})$ is given by the judgement $\Gamma \vdash^{\mathsf{n}}_{\mathcal{S}} a :_i A$ (or just $\Gamma \vdash^{\mathsf{n}} a{:_i}A$) for $i \in \mathbb{N}$ and defined by replacing the rule (**cofix**) from the typing rules of Figure 3 by the following one:*

$$(\mathbf{cofix}^n)\ \frac{\Gamma, x{:_{i+1}}A \vdash^{\mathsf{n}} b{:_i}A \quad \Gamma \vdash^{\mathsf{n}} A{:_i}\mathsf{type}}{\Gamma \vdash^{\mathsf{n}} (\mathsf{cofix}\ x{:}A.b){:_i}A}\ i \geq n$$

4. Basic Properties

In this section we prove some basic properties on CoPTSn's. which apply to CoPTS's as well since we have that $\Gamma \vdash a{:_i}A$ iff $\Gamma \vdash^0 a{:_i}A$.

Theorem 4.1 (Confluence)**.** *$(\mathcal{T}, \twoheadrightarrow_{\beta\sigma\gamma})$ is confluent.*

Proof. This follows from [35, Corollary 13.6] (see also [33]) by observing that $(\mathcal{T}, \twoheadrightarrow_{\beta\sigma\gamma})$ is an orthogonal combinatory reduction system. □

Theorem 4.2 (σ-strong normalization)**.** *Let $a \in \mathcal{C}$. Then, a is strongly σ-normalizing.*

Proof. Observe that the number of symbols decrease in each σ-reduction step. □

The notation Γ_{+k} means that we add k to the index of every hypothesis in Γ.

Theorem 4.3 (Time Adjustment)**.** *If $\Gamma, \Gamma' \vdash^{\mathsf{n}} a{:_i}A$ then $\Gamma, \Gamma'_{+k} \vdash^{\mathsf{n}} a{:_{i+k}}A$.*

The above theorem is proved by induction on the derivation.

Lemma 4.4 (Substitution)**.** *If $\Gamma \vdash^{\mathsf{n}} a{:_i}A$ and $\Gamma, x{:_i}A, \Gamma' \vdash^{\mathsf{n}} b{:_j}B$ then $\Gamma, \Gamma'[x := a] \vdash^{\mathsf{n}} b[x := a]{:_j}B[x := a]$.*

Proof. This lemma follows by induction on the derivation using Theorem 4.3 for the case of the (start)-rule. □

Theorem 4.5 (Subject Reduction)**.** *Let $a \twoheadrightarrow_{\beta\sigma\gamma} a'$. If $\Gamma \vdash^{\mathsf{n}} a{:_i}A$ then $\Gamma \vdash^{\mathsf{n}} a'{:_i}A$.*

Proof. We extend the reduction to contexts $\Gamma \to_{\beta\sigma\gamma} \Gamma'$ by allowing to reduce the types in Γ. We have to prove the following two statements simultaneously:

1. If $\Gamma \vdash^{\mathsf{n}} a{:_i}A$ and $a \twoheadrightarrow_{\beta\sigma\gamma} a'$ then $\Gamma \vdash^{\mathsf{n}} a'{:_i}A$.
2. If $\Gamma \vdash^{\mathsf{n}} a{:_i}A$ and $\Gamma \twoheadrightarrow_{\beta\sigma\gamma} \Gamma'$ then $\Gamma' \vdash^{\mathsf{n}} a{:_i}A$.

We use Lemma 4.4, Theorem 4.3 and the analogous of Generation Lemma [2, Lemma 5.2.13] adapted to the typing rules for CoPTS's □

Theorem 4.6 (Uniqueness of Types)**.** *Let $\mathcal{S}$ be single sorted. If $\Gamma \vdash^n a{:}_i A$ and $\Gamma \vdash^n a{:}_i A'$ then $A =_{\beta\sigma\gamma} A'$.*

The proof of the above theorem is similar to [2, Lemma 5.2.21].

Definition 4.7 (Strongly Normalizing CoPTS)**.** *Let ρ be a notion of reduction. We say that $\lambda^{\mathsf{co}}_{\mathsf{n}}(\mathcal{S})$ is strongly ρ-normalizing if for all $\Gamma \vdash^{\mathsf{n}} a{:}_i A$, we have that a and A are strongly ρ-normalizing.*

Notation 4.8. $\lambda^{\mathsf{co}}_{\mathsf{n}}(\mathcal{S}) \models \rho$*-SN if* $\lambda^{\mathsf{co}}_{\mathsf{n}}(\mathcal{S})$ *is strongly ρ-normalizing.*

We use the following abbreviations:

$$\bot = \Pi\alpha{:}\mathsf{type}.\alpha$$
$$\mathsf{S} = \lambda\alpha{:}\mathsf{type}.\Pi\beta{:}\mathsf{type}.(\alpha \to \beta \to \beta) \to \beta$$

We consider the context Γ_0 defined as $z_1{:}\bot$ where z_1 is a fresh variable.

Definition 4.9 (Encoding in $\lambda\omega$)**.** *Let $\Gamma \vdash^n d{:}_i D$. We define $\{d\}$ by induction on d.*

$$\begin{array}{rl}
\{x\} & = x \\
\{s\} & = s \\
\{\Pi x{:}A.B\} & = \Pi x{:}\{A\}.\{B\} \\
\{\lambda x{:}A.b\} & = \lambda x{:}\{A\}.\{b\} \\
\{(a\ b)\} & = (\{a\}\ \{b\}) \\
\{\bullet A\} & = \{A\} \\
\{\circ a\} & = \{a\} \\
\{(\mathsf{await}\ a)\} & = \{a\} \\
\{(\mathsf{Stream}\ A)\} & = \mathsf{S}\ \{A\} \\
\{(\mathsf{cons}\ a\ b)\} & = \lambda\beta{:}\mathsf{type}.\lambda f{:}A_0 \to \beta \to \beta. \\
 & \quad f\ \{a\}\ (\{b\}\ \beta\ f) \\
\{(\mathsf{hd}\ a)\} & = \{a\}\ A_0\ (\lambda x{:}A_0\lambda y{:}A_0.x) \\
\{(\mathsf{tl}\ a)\} & = \{a\}(\mathsf{S}\ A_0)\ (\lambda x{:}(\mathsf{S}\ A_0)\lambda y{:}\ (\mathsf{S}\ A_0).y) \\
\{(\mathsf{cofix}\ x{:}A.b)\} & = (\lambda x{:}\{A\}.\{b\})\ (z_1\ \{A\})
\end{array}$$

When d is either $(\mathsf{cons}\ a\ b)$, $(\mathsf{tl}\ a)$ or $(\mathsf{hd}\ a)$, we define the type A_0 as the β-normal form (if it exists) of $\{A\}$ where A is a type satisfying in each one of those cases:

$$\Gamma \vdash^n (\mathsf{cons}\ a\ b){:}_i(\mathsf{Stream}\ A)$$
$$\Gamma \vdash^n (\mathsf{tl}\ a){:}_i(\mathsf{Stream}\ A)$$
$$\Gamma \vdash^n (\mathsf{hd}\ a){:}_i A$$

The map $\{\}$ is extended to contexts in the obvious way.

$$\{x_1{:}_{i_1}A_1, \ldots, x_n{:}_{i_n}A_n\} = x_1{:}_{i_1}\{A_1\}, \ldots, x_n{:}_{i_n}\{A_n\}$$

Theorem 4.10. *Let $\mathcal{S}$ be singly sorted and $\lambda(\mathcal{S})$ be strongly β-normalizing. If $\Gamma \vdash^n_{\mathcal{S}} d :_i D$ then $\{\Gamma\}, \{d\}, \{D\}$ are well defined and $\Gamma_0, \{\Gamma\} \vdash_{\mathcal{S}} \{d\} : \{D\}$.*

Proof. This follows by induction on the structure of the term using Generation Lemma. We show the case $d = (\mathsf{cons}\ a\ b)$. Suppose $\Gamma \vdash^n (\mathsf{cons}\ a\ b){:}_i(\mathsf{Stream}\ A)$ and $\Gamma \vdash^n (\mathsf{cons}\ a\ b){:}(\mathsf{Stream}\ A')$. It follows from Theorem 4.6 that $A =_{\beta\sigma\gamma} A'$. By Theorem 4.11, we have that $\{A\} =_\beta \{A'\}$. We have that $\Gamma \vdash^n a : A$ and $\Gamma \vdash^n a :_i A'$. By Induction Hypothesis, $\{\Gamma\} \vdash^n \{a\} : \{A\}$ and $\{\Gamma\} \vdash^n \{a\} : \{A'\}$. Since $\lambda(S)$ is strongly β-normalizing, A_0 from Definition 4.9 is uniquely determined since the β-normal forms of A and A' are the same. Hence, $\{d\}$ is well defined. □

The following statements are not difficult to prove.

Theorem 4.11. *1. If $a \to_\beta a'$ then $\{a\} \to_\beta \{a'\}$.*
2. If $a \to_\sigma a'$ then $\{a\} \to_\beta^= \{a'\}$, i.e. either $\{a\} \to_\beta \{a'\}$ or $\{a\}{=}\{a'\}$.

Theorem 4.12 (Preservation of Strong Normalization without Contracting Fixpoints)**.** *Let $\mathcal{S}$ be single sorted such that $\lambda(\mathcal{S})$ extends $\lambda\omega$. If $\lambda(\mathcal{S}) \models \beta$-SN then $\lambda^{\mathsf{co}}_{\mathsf{n}}(\mathcal{S}) \models \beta\sigma$-SN.*

Proof. Suppose $\Gamma \vdash a :_i A$. By Theorem 4.10, we have that $\{a\}$ is typable in $\lambda(\mathcal{S})$ and hence, it is β-strongly normalizing. We prove that a is strongly $\beta\sigma$-normalizing by contradiction. Suppose that a is not strongly $\beta\sigma$-normalizing. That is, suppose there exists an infinite $\beta\sigma$-reduction sequence starting from a. Observe that the number of β-reduction steps in this sequence must be infinite because σ is strongly normalizing (Theorem 4.2). Hence, the sequence is of the form:

$$a = a_0 \twoheadrightarrow_\sigma a_1 \to_\beta a_2 \twoheadrightarrow_\sigma a_3 \to_\beta a_4 \twoheadrightarrow_\sigma a_5 \to_\beta a_6 \ldots$$

By Theorem 4.11, we have that:

$$\{a\} = \{a_0\} \twoheadrightarrow_\beta \{a_1\} \to_\beta \{a_2\} \twoheadrightarrow_\beta \{a_3\} \to_\beta \{a_4\} \twoheadrightarrow_\beta \ldots$$

which contradicts the fact that $\{a\}$ is β-strongly normalizing. □

Corollary 4.13. *$\lambda^{\mathsf{co}}(EC)$ and all the systems of the λ-cube extended with corecursion are strongly $\beta\sigma$-normalizing.*

Proof. It follows from Theorems 2.15 and 4.12, that $\lambda^{\mathsf{co}}(EC)$ is strongly $\beta\sigma$-normalizing. All the systems of the λ-cube extended with corecursion are strongly $\beta\sigma$-normalizing because they are all included in $\lambda^{\mathsf{co}}(EC)$. □

Remark 4.14. *The previous theorem is about $\beta\sigma$-reduction and does not mention γ for the reason , that CoPTS's can not be γ-normalizing, as terms containing a fixed point may have an infinite γ-reduction.*

5. Infinite Pseudoterms

The program fib of Example 3.8 is not finitary but infinitary normalizing, i.e. the normal form of fib is an infinite term of the form

$$(\mathsf{cons}\ 1\ (\mathsf{cons}\ 1\ (\mathsf{cons}\ 2\ (\mathsf{cons}\ 3\ (\mathsf{cons}\ 5\ \ldots)))))$$

What do the infinite normal forms of typable terms look like? We cannot type $(\mathsf{cofix}\ x{:}A.f\ x)$ but we can type the term $(\mathsf{cofix}\ x{:}A.f\ (\circ x))$ as follows.

$$A :_i \mathsf{type}, f :_i \bullet A \to A \vdash (\mathsf{cofix}\ x{:}A.f\ (\circ x)) :_i A$$

To describe the infinite normal forms of the typable terms, we define a set $\mathcal{C}^\infty$ of finite and infinite terms as a metric completion using an appropriate metric. We do not use any of the existing metrics defined in the literature for infinitary term rewriting systems, infinitary lambda calculus or infinitary combinatory reduction systems [28, 29, 32]. This is because we want to connect the modal operator, the level n of a CoPTSn and the index i in $\Gamma \vdash a{:}_i A$ to the depth (Theorems 6.4 and 7.6). This connection is used in the proofs on infinitary weak and strong normalization.

Definition 5.1 (Subterm at position p)**.** *Let p be a sequence of 0's and 1's. The subterm at position p, denoted as $a|_p$, is defined by induction as follows.*

$$\begin{array}{rlrl}
a|_\epsilon & = a & & \\
(\Pi x{:}A.B)|_{0.p} & = A|_p & (\Pi x{:}A.B)|_{1.p} & = B|_p \\
\lambda x{:}A.b|_{0.p} & = A|_p & \lambda x{:}A.b|_{1.p} & = b|_p \\
(a\ b)|_{0.p} & = a|_p & (a\ b)|_{1.p} & = b|_p \\
(\bullet A)|_{0.p} & = A|_p & & \\
(\circ a)|_{0.p} & = a|_n & & \\
((\mathsf{await}\ a))|_{0.p} & = a|_p & & \\
(\mathsf{Stream}\ A)|_{0.p} & = A|_p & & \\
(\mathsf{cons}\ a\ b)|_{0.p} & = a|_p & (\mathsf{cons}\ a\ b)|_{1.p} & = b|_p \\
(\mathsf{hd}\ a)|_{0.p} & = a|_p & & \\
(\mathsf{tl}\ a)|_{0.p} & = a|_p & & \\
(\mathsf{cofix}\ x{:}A.b)|_{0.p} & = A|_p & (\mathsf{cofix}\ x{:}A.b)|_{1.p} & = b|_p
\end{array}$$

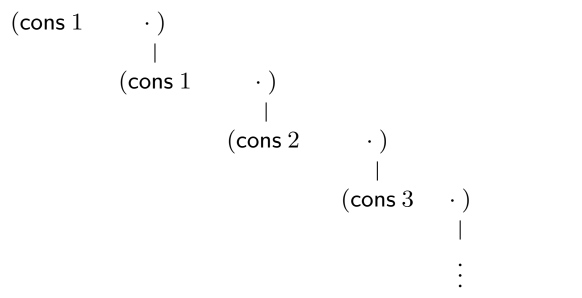

Figure 4. The term $\mathsf{partialfib}$ represented as a tree

Let $\mathsf{partialfib} = (\mathsf{cons}\ 1\ (\mathsf{cons}\ 1\ (\mathsf{cons}\ 2\ (\mathsf{cons}\ 3\ (\mathsf{cons}\ 5\ \mathsf{fib})))))$ be the result of unfolding fib three times. The subterm of partialfib at position $1.1.1$ is $(\mathsf{cons}\ 3\ (\mathsf{cons}\ 5\ \mathsf{fib}))$.

Let p, q be two positions. We define $p < q$ if there exists a position r such that $q = p.r$.

Definition 5.2 (Depth). *The depth of a subterm b of a is the number of subterms of a at positions $q < p$ such that $a|_q$ is either of the form* $(\mathsf{cons}\ c\ d)$ *or* $(\circ c)$.

For example, the depth of $(\mathsf{cons}\ 3\ (\mathsf{cons}\ 5\ \mathsf{fib}))$ in partialfib is three. Figure 4 illustrates our notion of depth by drawing the term as a finitely branched tree.

We need to define the notion of truncations. The result of a truncation is a pseudoterm that may contain a special constant $\perp$.

Definition 5.3 (Truncation). *The truncation of a at depth n is denoted by a^n and it is defined as the result of replacing the subterms of a at depth n by $\perp$. Equivalently, it can be defined by induction as follows.*

$$\begin{aligned}
a^0 &= \perp \\
x^{n+1} &= x \\
s^{n+1} &= s \\
\Pi x{:}A.B^{n+1} &= \Pi x{:}A^{n+1}.B^{n+1} \\
\lambda x{:}A.b^{n+1} &= \lambda x{:}A^{n+1}.b^{n+1} \\
(b\ a)^{n+1} &= (a^{n+1}\ b^{n+1}) \\
(\bullet A)^{n+1} &= \bullet A^{n+1} \\
(\circ a)^{n+1} &= \circ a^{n} \\
((\mathsf{await}\ a))^{n+1} &= (\mathsf{await}\ a^{n+1}) \\
(\mathsf{Stream}\ A)^{n+1} &= (\mathsf{Stream}\ A^{n+1}) \\
(\mathsf{cons}\ a\ b)^{n+1} &= (\mathsf{cons}\ a^{n+1}\ b^{n}) \\
(\mathsf{hd}\ a)^{n+1} &= (\mathsf{hd}\ a^{n+1}) \\
(\mathsf{tl}\ a)^{n+1} &= (\mathsf{tl}\ a^{n+1}) \\
(\mathsf{cofix}\ x{:}A.b)^{n+1} &= (\mathsf{cofix}\ x{:}A^{n+1}.b^{n+1})
\end{aligned}$$

For example, the truncation of partialfib at depth three is

$$(\mathsf{cons}\ 1\ (\mathsf{cons}\ 1\ (\mathsf{cons}\ 2\ \perp)))$$

Definition 5.4 (Metric). *We define a metric $d : \mathcal{C} \times \mathcal{C}_\perp \to [0,1]$ as follows: $d(a,b) = 0$, if $a = b$ and $d(a,b) = 2^{-m}$, where $m = max\{a^n = b^n \mid n \in \mathsf{Nat}\}$.*

Note that $(\mathcal{C}, d)$ is an ultrametric space.

Definition 5.5 (Set of Finite and Infinite Pseudoterms). *$\mathcal{C}^\infty$ is the metric completion of $(\mathcal{C}, d)$.*

It should be noted that restricted to the subset of lambda terms in $\mathcal{C}$ the metric d is in fact the discrete metric. The above definition excludes many infinite terms.

Example 5.6 (Infinite Pseudoterms in $\mathcal{C}^\infty$). *The following are infinite pseudoterms belonging to $\mathcal{C}^\infty$:*

$(\mathsf{cons}\ 0\ (\mathsf{cons}\ 0\ \ldots))$	$\circ(\circ(\circ\ldots))$
$(\mathsf{cons}\ 0\ (\mathsf{tl}\ (\mathsf{cons}\ 0\ (\mathsf{tl}\ \ldots))))$	$(\mathsf{await}\ (\circ(\mathsf{await}\ (\circ\ldots))))$

The set $\mathcal{C}^\infty$ is strictly included in any of the sets of finite and infinite terms defined for infinitary term rewriting systems, infinitary lambda calculus and infinitary combinatory reduction systems [28, 29, 32]. This is shown in the following example:

Example 5.7 (What is not in $\mathcal{C}^\infty$?). *The following "terms" do not belong to $\mathcal{C}^\infty$.*

$$\begin{array}{l}
(f\ (f\ \ldots)) \\
\lambda x_1.\lambda x_2.\lambda x_3.\ldots \\
(((\ldots)x_3)x_2)x_1 \\
(\mathsf{tl}\ (\mathsf{tl}\ (\mathsf{tl}\ \ldots))) \\
(\mathsf{await}\ (\mathsf{await}\ (\mathsf{await}\ \ldots)))
\end{array}$$

The first three terms are characteristic examples of respectively a Böhm tree, Lévy Longo and Berarducci tree [1, 3, 6, 28, 39, 40]. These terms belong to three increasingly larger Cauchy completions of the set of finite lambda terms. These three completion can be constructed by using the '001' metric, the '101' metric and the '111'metric respectively. The first completion using the '001' metric, which is a subset of the completion obtained with the '101' metric, which in turn is contained in the Cauchy completion made with the '111' metric. Each of these 'xyz' completions is closed under 'xyz'-strongly converging reduction [28, 29].

The last two terms belong to the metric completions defined for infinitary term rewriting and infinitary combinatory reduction systems [28, 32].

Notation 5.8 (Reduction at depth n). *We denote $a \xrightarrow{n}_\rho b$ if the contracted ρ-redex is at depth n.*

Definition 5.9 (Strongly Converging Reductions). *A strongly convergent ρ-reduction sequence of length α (an ordinal) is a sequence $\{a_\beta \mid \beta \leq \alpha\}$ of terms in $\mathcal{C}^\infty$, such that*

1. *$a_\beta \to_\rho a_{\beta+1}$ for all $\beta < \alpha$,*
2. *$a_\lambda = \lim_{\beta<\lambda} a_\beta$ for every limit ordinal $\lambda \leq \alpha$.*
3. *$\lim_{i\to\lambda} d_i = \infty$ where d_i is the depth of the redex contracted at $a_i \to_\rho a_{i+1}$.*

Notation 5.10 (Strongly convergent reduction). *$a \twoheadrightarrow\!\!\!\!\!\rightarrow_\rho b$ denotes a strongly converging reduction from a to b.*

By construction, the collection of finite and pseudoterms $\mathcal{C}^\infty$ is closed under strongly converging reduction.

Example 5.11 (Strongly Converging Reductions). *We have that* $\mathsf{zeros} \twoheadrightarrow\!\!\!\!\!\rightarrow_\gamma (\mathsf{cons}\ 0\ (\mathsf{cons}\ 0\ (\mathsf{cons}\ 0\ \ldots)))$ *via the following strongly convergent reduction of length ω (we indicate the depth of the contracted redex in the superscipt of the rewrite arrows):*

$$\begin{array}{lll}
\mathsf{zeros} & \xrightarrow{0}_\gamma & (\mathsf{cons}\ 0\ \mathsf{zeros}) \\
 & \xrightarrow{1}_\gamma & (\mathsf{cons}\ 0\ (\mathsf{cons}\ 0\ \mathsf{zeros})) \\
 & \xrightarrow{2}_\gamma & (\mathsf{cons}\ 0\ (\mathsf{cons}\ 0\ (\mathsf{cons}\ 0\ \mathsf{zeros}))) \\
 & \xrightarrow{3}_\gamma & \ldots \\
 & \vdots & \\
 & & (\mathsf{cons}\ 0\ (\mathsf{cons}\ 0\ (\mathsf{cons}\ 0\ \ldots)))
\end{array}$$

Let $\mathsf{nfzeros} = (\mathsf{cons}\ 0\ (\mathsf{cons}\ 0\ (\mathsf{cons}\ 0\ \ldots)))$ *be the infinite normal form of* zeros. *We show an example of a reduction sequence*

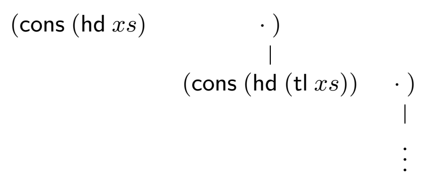

Figure 5. Infinite normal form of (map Nat Nat id xs) as a tree

of length ω^2.

$$\begin{array}{lll} \mathsf{zeross} & \twoheadrightarrow\!\!\!\!\twoheadrightarrow_\gamma & (\mathsf{cons\ nfzeros\ zeross}) \\ & \twoheadrightarrow\!\!\!\!\twoheadrightarrow_\gamma & (\mathsf{cons\ nfzeros\ (cons\ nfzeros\ zeross)}) \\ & \twoheadrightarrow\!\!\!\!\twoheadrightarrow_\gamma & \ldots \\ \vdots & & \\ & & (\mathsf{cons\ nfzeros\ (cons\ nfzeros\ (\ldots))}) \end{array}$$

As we mentioned in the introduction, there exists a strongly converging reduction sequence of length ω *from* zeross *to the infinite normal form by following a depth-first-leftmost strategy.*

Example 5.12 (Non-strongly Converging Reductions)**.** *The following infinite reduction sequences are not strongly convergent:*

$$\Omega \xrightarrow{0}_\gamma \Omega \xrightarrow{0}_\gamma \Omega \xrightarrow{0}_\gamma \ldots$$

$$\Omega_{\mathsf{tail}} \xrightarrow{0}_\gamma (\mathsf{tl}\ \Omega_{\mathsf{tail}}) \xrightarrow{0}_\gamma (\mathsf{tl}\ (\mathsf{tl}\ \Omega_{\mathsf{tail}})) \xrightarrow{0}_\gamma \ldots$$

$$\begin{array}{lll} (\mathsf{cofix}\ x{:}A.f\ x) & \xrightarrow{0}_\gamma & f\ (\mathsf{cofix}\ x{:}A.f\ x) \\ & \xrightarrow{0}_\gamma & f\ (f\ (\mathsf{cofix}\ x{:}A.f\ x)) \\ & \xrightarrow{0}_\gamma & \ldots \end{array}$$

6. Infinitary Weak Normalization

In this section, we introduce the concept of infinitary weakly normalizing typing system. We prove that a CoPTS is infinitary weakly $\beta\sigma\gamma$-normalizing provided that the original PTS is β-strongly normalizing. Proving infinitary weak normalization poses several difficulties:

1. Contracting γ-redexes can create $\beta\sigma$-redexes.
2. Contracting $\beta\sigma$-redexes may decrease the depth of any subterm.

We overcame these difficulties by using the auxiliary system $\vdash^n$.

Definition 6.1 (Infinitary Weak Normalization)**.** *Let* ρ *be a notion of reduction. We say that* a *is infinitary weakly* ρ*-normalizing if there exists a* ρ*-normal form* b *such that* $a \twoheadrightarrow\!\!\!\!\twoheadrightarrow_\rho b$.

The undesirable terms (see Example 3.9) are not infinitary weakly $\beta\sigma\gamma$-normalizing. The term (mapNat Nat id xs) is weakly $\beta\sigma\gamma$-normalizing. Its normal form is depiced as a tree in Figure 5. Our tree representation reflects the notion of depth.

Definition 6.2 (Infinitary Weak Normalizing CoPTS)**.** *We say that* $\lambda^{\mathsf{co}}(\mathcal{S})$ *is infinitary weakly* ρ*-normalizing if for all* $a \in \mathcal{C}$ *such that* $\Gamma \vdash a :_i A$*, we have that* a *is infinitary weakly* ρ*-normalizing.*

Notation 6.3. $\lambda^{\mathsf{co}}(\mathcal{S}) \models \rho\text{-}WN^\infty$ *if* $\lambda^{\mathsf{co}}(\mathcal{S})$ *is infinitary weakly* ρ*-normalizing*

In the next theorem, we relate the n of a CoPTSn with the truncation at depth n.

Theorem 6.4 (Truncation at depth n of a term in CoPTSn)**.** *Let* $n \geq i$*. If* $\Gamma \vdash^n a{:}_iA$ *then* a^{n-i} *is in* γ*-normal form, i.e.* a^{n-i} *does not have fixed points.*

Proof. We prove it simmultaneously with the statement: if $x{:}_jB$ is in Γ then B^{n-j} is in γ-normal form. □

We define a function that contracts all cofix occurrences of a pseudoterm just once.

Definition 6.5. *We define* $\lceil a \rceil$ *by induction on* a.

$$\begin{array}{ll} \lceil x \rceil & = x \\ \lceil s \rceil & = s \\ \lceil \Pi x{:}A.b \rceil & = \Pi x{:}\lceil A \rceil.\lceil b \rceil \\ \lceil \lambda x{:}A.b \rceil & = \lambda x{:}\lceil A \rceil.\lceil b \rceil \\ \lceil (a\ b) \rceil & = (\lceil a \rceil\ \lceil b \rceil) \\ \lceil \bullet A \rceil & = \bullet\lceil A \rceil \\ \lceil \circ a \rceil & = \circ\lceil a \rceil \\ \lceil (\mathsf{await}\ a) \rceil & = (\mathsf{await}\ \lceil a \rceil) \\ \lceil (\mathsf{Stream}\ A) \rceil & = (\mathsf{Stream}\ \lceil A \rceil) \\ \lceil (\mathsf{cons}\ a\ b) \rceil & = (\mathsf{cons}\ \lceil a \rceil\ \lceil b \rceil) \\ \lceil (\mathsf{hd}\ a) \rceil & = (\mathsf{hd}\ \lceil a \rceil) \\ \lceil (\mathsf{tl}\ a) \rceil & = (\mathsf{tl}\ \lceil a \rceil) \\ \lceil (\mathsf{cofix}\ x{:}A.b) \rceil & = \lceil b \rceil[x := (\mathsf{cofix}\ x{:}\lceil A \rceil.\lceil b \rceil)] \end{array}$$

The map $\lceil\rceil$ is extended to contexts in the obvious way.

$$\lceil x_1 :_{i_1} A_1, \ldots, x_n :_{i_n} A_n \rceil = x_1 :_{i_1} \lceil A_1 \rceil, \ldots, x_n :_{i_n} \lceil A_n \rceil$$

Note that $a \twoheadrightarrow\!\!\!\!\twoheadrightarrow_\gamma \lceil a \rceil$.

Theorem 6.6. *Let* $\Gamma \vdash^n a{:}_iA$*. Then* $\lceil \Gamma \rceil \vdash^{n+1} \lceil a \rceil{:}_i\lceil A \rceil$.

Proof. This is proved by induction on the derivation. We show the key case:

$$(\mathbf{cofix}^n)\ \frac{\Gamma, x{:}_{i+1}A \vdash^n b{:}_iA \quad \Gamma \vdash^n A{:}_i\mathsf{type}}{\Gamma \vdash^n (\mathsf{cofix}\ x{:}A.b){:}_iA}\ i \geq n$$

By Induction Hypothesis,

$$\lceil \Gamma \rceil, x{:}_{i+1}\lceil A \rceil \vdash^{n+1} \lceil b \rceil{:}_i\lceil A \rceil \tag{1}$$

$$\lceil \Gamma \rceil \vdash^{n+1} \lceil A \rceil{:}_i\mathsf{type} \tag{2}$$

From the above rule, we know that $i \geq n$. However, we cannot apply $\mathbf{cofix}^{n+1}$ unless $i \geq n+1$. The trick is to apply Time Adjustment (Theorem 4.3) to (1) and (2).

$$\begin{array}{l} \lceil \Gamma \rceil, x{:}_{i+2}\lceil A \rceil \vdash^{n+1} \lceil b \rceil{:}_{i+1}\lceil A \rceil \\ \lceil \Gamma \rceil \vdash^{n+1} \lceil A \rceil{:}_{i+1}\mathsf{type} \end{array}$$

Since $i+1 \geq n+1$, we can apply ($\mathbf{cofix}^{n+1}$) and obtain:

$$\lceil \Gamma \rceil \vdash^{n+1} (\mathsf{cofix}\ x{:}\lceil A \rceil.\lceil b \rceil){:}_{i+1}\lceil A \rceil \tag{3}$$

It follows from Substitution Lemma (Lemma 4.4), (1) and (3) that

$$\lceil \Gamma \rceil \vdash^{n+1} \lceil b \rceil[x := (\mathsf{cofix}\ x{:}\lceil A \rceil.\lceil b \rceil)]{:}_i\lceil A \rceil$$

Since $\lceil (\mathsf{cofix}\ x{:}A.b) \rceil = \lceil b \rceil[x := (\mathsf{cofix}\ x{:}\lceil A \rceil.\lceil b \rceil)]$, we are done. □

Theorem 6.7 (Infinitary Weak $\beta\sigma\gamma$-Normalization)**.** *Let* $\mathcal{S}$ *be singly sorted such that* $\lambda(\mathcal{S})$ *extends* $\lambda\omega$*. If* $\lambda(\mathcal{S}) \models \beta$*-SN then* $\lambda^{\mathsf{co}}(\mathcal{S}) \models \beta\sigma\gamma\text{-}WN^\infty$*. Moreover, the infinitary weak* $\beta\sigma\gamma$*-normal forms are obtained in* ω*-steps.*

Proof. Suppose $\Gamma \vdash a :_i A$. Hence, $\Gamma \vdash^0 a :_i A$. We show that there exists a normalizing strategy starting from a. We construct a reduction sequence of following form:

$$a = a_0 \twoheadrightarrow_\gamma a_0' \twoheadrightarrow_{\beta\sigma} a_1 \twoheadrightarrow_\gamma a_1' \twoheadrightarrow_{\beta\sigma} a_2 \ldots \tag{4}$$

We define a_0' as $\lceil a_0 \rceil$. By Theorem 6.6, we have that $\lceil \Gamma \rceil \vdash^1 a_0' :_i \lceil A \rceil$. We define a_1 as the $\beta\sigma$-normal form of a_0' which exists by Theorem 4.12. By Theorem 4.5, $\lceil \Gamma \rceil \vdash^1 a_1 :_i \lceil A \rceil$. We repeat this process for each n. The reduction sequence (4) has the following form:

$$a = a_0 \twoheadrightarrow_{\beta\sigma\gamma} a_1 \twoheadrightarrow_{\beta\sigma\gamma} a_2 \twoheadrightarrow_{\beta\sigma\gamma} \dots \qquad (5)$$

where for all n there exist Γ_n and A_n such that $\Gamma_n \vdash^n a_n :_i A_n$. By Theorem 6.4, we have that $(a_n)^n$ is in $\beta\sigma\gamma$-normal form for all $n \geq i$. From i onwards, the sequence of truncations $a_i^0, a_{i+1}^1, a_{i+2}^2, \dots$ is increasing (with respect to the subterm relation). It is clear that the reduction sequence (5) is strongly converging. Its limit a_ω exists and it is in infinite $\beta\sigma\gamma$-normal form. □

Corollary 6.8. *$\lambda^{\text{co}}(EC)$ and all the systems of the λ-cube extended with corecursion are infinitary weakly $\beta\sigma\gamma$-normalizing.*

Proof. It follows from Theorems 2.15 and 6.7, that $\lambda^{\text{co}}(EC)$ is infinitary weakly $\beta\sigma\gamma$-normalizing. Since all the systems of the λ-cube extended with corecursion are included in $\lambda^{\text{co}}(EC)$, we can conclude infinitary weakly $\beta\sigma\gamma$-normalization for all of them. □

7. Infinitary Strong Normalization

In this section, we connect the index and the modality with the depth. We also define the concept of infinitary strong normalization and prove that CoPTS's are strongly γ-normalizing.

Definition 7.1 (Infinitary Strong Normalization)**.** *Let ρ be a notion of reduction. We say that a is infinitary strongly ρ-normalizing if we have that all ρ-reduction sequences starting from a are strongly convergent.*

For example, the term $(\lambda x{:}A.\mathsf{zeros})\Omega$ is infinitary weakly $\beta\sigma\gamma$-normalizing but it is not infinitary strongly $\beta\sigma\gamma$-normalizing.

Definition 7.2 (Infinitary Strongly Normalizing CoPTS)**.** *Let ρ be a notion of reduction. We say that $\lambda^{\text{co}}(\mathcal{S})$ is infinitary strongly ρ-normalizing if for all $a \in \mathcal{C}$ such that $\Gamma \vdash^{\text{co}} a :_i A$ we have that a is strongly ρ-normalizing.*

Notation 7.3. *$\lambda^{\text{co}}(\mathcal{S}) \models \rho$-$SN^\infty$ if $\lambda^{\text{co}}(\mathcal{S})$ is infinitary strongly ρ-normalizing*

Note that $\lambda^{\text{co}}(\mathcal{S}) \models \rho\text{-SN}^\infty$ implies $\lambda^{\text{co}}(\mathcal{S}) \models \rho\text{-WN}^\infty$.

Theorem 7.4 (Depth of Variables)**.** *Let $\Gamma, x{:}_iA, \Gamma' \vdash b{:}_jB$. Then the depth of all occurrences of x in b is greater than $i - j$ if $i > j$.*

Proof. We have to prove it simmultaneosuly with the statement: if $\Gamma, x{:}_iA, \Gamma' \vdash b{:}_jB$ and $y{:}_kC \in \Gamma'$ then all occurrences of x in C occur at depth greater than $i - k$ if $i > k$. □

Corollary 7.5 (Depth of x of type $(\bullet A)$)**.** *If $\Gamma, x{:}_i(\bullet A) \vdash b{:}_iB$ then the depth of all occurrences of x in b is greater than 0.*

Corollary 7.6 (Depth of x in cofix)**.** *If $\Gamma \vdash (\mathsf{cofix}\ x{:}A.b){:}_iA$ then the depth of all occurrences of x in b is greater than 0.*

As a consequence of Theorem 7.4, we have that if a fixed point occurs in a typable term at depth n then it will occur at depth $n+1$ after its contraction. Let $\mathsf{cardfix}_n(a)$ be the number of fixed points of a at depth n.

Theorem 7.7 (Strong Normalization of γ-reduction at depth n)**.** *Let $\Gamma \vdash a{:}_iA$.*

1. *If $a \overset{m}{\to}_\gamma b$ and $m \geq n$ then $\mathsf{cardfix}_n(a) > \mathsf{cardfix}_n(b)$.*
2. *Any reduction sequence of $\overset{n}{\to}_\gamma$ steps is finite.*

Proof. The first statement is proved by induction on the structure of a using Corollary 7.6. The second one follows by absurd. Suppose there is an infinite reduction sequence starting from $a = a_0 \overset{n}{\to}_\gamma a_1 \overset{n}{\to}_\gamma a_2 \dots$. From the first part, we would have an infinite decreasing sequence of natural numbers $\mathsf{cardfix}(a_0) > \mathsf{cardfix}(a_1) > \dots$ This is a contradiction. This means that this reduction sequence has to be finite. □

Theorem 7.8 (Infinitary Strong γ-normalization)**.** *We have that $\lambda^{\text{co}}(\mathcal{S}) \models \gamma$-$SN^\infty$.*

Proof. Suppose there is an infinite reduction sequence starting from $a = a_0 \to_\gamma a_1 \to_\gamma a_2 \dots$ with an infinite number of steps at depth 0. By Theorem 7.7, the number of steps at depth 0 in that sequence should be finite. Hence, there exists a_k such that from a_k onwards, all reduction steps contract redexes at depth greater than 1. We repeat the process for $n = 1$ and then for each depth n observing that the number of fixed points of a term at depth n decreases if we only contract redexes at depth greater or equal than n. □

8. Conclusions and Related Work

Our normalization result (Theorem 6.7) says that the terms typable in a CoPTS have a possible infinite normal form that is an element of the set $\mathcal{C}^\infty$. Restricted to lambda terms, the equality relation induced by $\beta\sigma\gamma$-normal form —two terms are equivalent iff they have the same infinite normal form — is a strict subrelation of the equality relations induced the notions of Böhm, Lévy Longo and Berarducci trees.

Comparision with other typed lambda calculi. Nakano defines a typed lambda calculus with modality, subtyping and recursive types where Curry's and Turing's fixed point combinators Y and Θ can be typed and both have type $(\bullet A \to A) \to A$ [44]. Nakano proves that all typable terms have a Böhm tree without $\perp$ which amounts to saying that they have an infinite β-normal form in the infinitary lambda calculus with the '001' metric of [29]. Nakano's type system can type terms that CoPTS's cannot type (their infinite normal forms do not belong to $\mathcal{C}^\infty$). For example, it can type $\mathsf{Y}f$ whose infinite normal form is the following:

$$(f\ (f\ \dots))$$

and also $\mathsf{Y}(\lambda xy.yx)$ is typable using the recursive type $\mu X.(\bullet X \to B) \to B$ whose infinite normal form is the following:

$$(\lambda y_1.y_1(\lambda y_2.y_2(\lambda y_3.y_3 \dots)))$$

Krishnaswami and Benton's typed lambda calculus of reactive programs use an equational theory instead of reduction [36]. Corollary 4.13 generalizes and strengthens in several directions the result in [36] where only weak normalization is proved for the fragment of $\lambda^{\text{co}}_{\to}$ without fixed points.

Krishnaswami, Benton and Hoffman show another variant of $\lambda^{\text{co}}_{\to}$ with linearity in [38]. They define a notion of reduction and show that all typable terms reduce to some value. Values are essentially abstractions meaning that this result is somewhat similar to weak head normalization.

Giménez studies an extension of the calculus of constructions with inductive and coinductive types [21]. A type constructor $\widehat{A}$ is introduced that resembles a modal operator. The meaning of this operator is not the same as $\bullet A$. While $\bullet A$ can be understood as the information displayed in the *future*, $\widehat{A}$ represents the set of terms that are *guarded* by constructors.

Borghuis studies modal pure type systems (MPTS's) in [9]. CoPTS's are essentially MPTS's with fixed points and streams but without the double negation axiom. The contexts for MPTS's look a bit different because they group together type declarations with the same index $\Gamma_n, \Gamma_{n-1}, \dots, \Gamma_0$ where $\Gamma_i = \{x_1{:}_iA_1, \dots, x_n{:}_iA_n\}$. Judgements in a MPTS can only infer types at time 0.

Other approaches to corecursion. Hutton and Jaskelioff studies the example of zeros'' and proposes a methodology that ensures that the fixed point of a function on streams is well defined [26]. On one hand, while they propose a methodology that each particular case has to be treated on its own way, the approach with typing treats all programs in "a uniform way" and could be automatized. On the other hand, Hutton and Jaskelioff can include functions such as zeros''' that CoPTS's cannot type.

$$\begin{aligned}\mathsf{zeros}''' = \quad & (\mathsf{cofix}\, xs{:}(\mathsf{Stream\ Nat}).) \\ & (\mathsf{cons}\ 0\ (\mathsf{interleave}\ xs\ (\mathsf{tl}\ xs)))\end{aligned}$$

The infinite normal form of zeros''' is

$$(\mathsf{cons}\ 0\ (\mathsf{cons}\ 0\ (\mathsf{cons}\ 0\ \ldots)))$$

Endrullis et al and Zantema et al use term rewriting systems to ensure the well-definedness of corecursive equations [16, 54]. The set of finite and infinite terms is defined as a map (or labelled tree) from positions to symbols. This set is equivalent to the ones defined in [28] as a metric completion and includes terms such as $(\mathsf{tl}\ (\mathsf{tl}\ (\mathsf{tl}\ \ldots)))$. The notion of productivity is defined as weak normalization but excluding terms that do not contain constructors such as the one shown above.

O'Connor implemented the fixed point operator cofix'' of type $(\bullet A \to A) \to A$ in Haskell.

Techniques to prove normalization. In order to prove preservation of strong normalization without contracting fixed points (Theorem 4.12), we use a translation from $\lambda^{\mathsf{co}}(\mathcal{S})$ into $\lambda\omega$. The technique using translations has been used before in the literature. For example, in [24] to prove that $\lambda_{\to} \models \beta$-SN implies $\lambda P \models \beta$-SN , or in [18] to prove that $\lambda\omega \models \beta$-SN implies $\lambda C \models \beta$-SN. To get preservation of strong normalization, the translation usually has to preserve reduction in a way that one step is mapped into one or more steps. In our case, the translation can cancel σ-steps. In spite of this, we can prove preservation of strong normalization using the fact that σ is strongly normalizing on untyped terms. A similar technique has been used to prove $\lambda\mathcal{S} \models \beta$-SN implies $\lambda^{\delta}(\mathcal{S}) \models \beta\delta$-SN where $\lambda^{\delta}(\mathcal{S})$ is the extension of $\lambda(\mathcal{S})$ with definitions where the translation can cancel δ-steps [48].

In order to prove infinitary weak normalization we used an auxiliary system $\vdash^n$ (and the unfolding $\lceil a \rceil$). This technique is used in [37] to prove that all typable terms have an m-normal form for a calculus based on $\lambda^{\mathsf{co}}_{\to}$ with linearity. The notion of m-normal form is defined in [37] in terms of the auxiliary system $\vdash^n$. This does not ensure yet that typable terms are well-behaved.

9. Future Work

It will be interesting to define a Böhm model for corecursion on streams by interpreting terms as infinite normal forms [6, 27]. However, to ensure that the model is well defined, we need to prove infinitary confluence besides infinitary weak normalization. The problem is that $\twoheadrightarrow\!\!\!\!\rightarrow_{\beta\sigma\gamma}$ is not confluent on untypable terms. We construct some counter-examples from the σ-rules which are hypercollapsing, i.e. they are of the form $C[x] \to x$ [28, 29].

Example 9.1 (Failure of Confluence)**.** *We have that*

$$\begin{tikzcd} & \Omega' \arrow[dl, "\sigma"'] \arrow[dr, two heads, "\gamma"] & \\ \Omega & & (\mathsf{tl}\ (\mathsf{cons}\ 0\ (\mathsf{tl}\ (\mathsf{cons}\ 0\ \ldots)))) \end{tikzcd}$$

cannot be joined. The terms Ω *and* $(\mathsf{tl}\ (\mathsf{cons}\ 0\ (\mathsf{tl}\ (\mathsf{cons}\ 0\ \ldots))))$ *can only reduce to themselves. Similarly,*

$$\begin{tikzcd} & \Omega'' \arrow[dl, "\sigma"'] \arrow[dr, two heads, "\gamma"] & \\ \Omega & & \mathsf{await}(\circ(\mathsf{await}(\circ(\ldots)))) \end{tikzcd}$$

The terms Ω *and* $\mathsf{await}(\circ(\mathsf{await}(\circ(\ldots))))$ *can only reduce to themselves.*

Ketema and Simonsen prove confluence up to hypercollapsing terms for orthogonal infinitary combinatory reduction systems [32]. However, we cannot apply their result. This is because $\mathcal{C}^{\infty}$ is strictly included in their syntax and their confluence result may give us a common reduct which is outside our syntax

We have proved that $\lambda^{\mathsf{co}}(\mathcal{S})$ is infinitary strongly γ-normalizing. However, it remains open to prove it for $\beta\sigma\gamma$.

Once we have a Böhm Model for corecursion on streams, we would like to find a way of integrating the syntactic model of Böhm trees which is an ultrametric space with the model of ultrametric spaces [7, 36] and the topos of trees [8]. The notion of contractive function should correspond to the notion of head normal form in the Böhm model.

In this paper we have considered only streams which is one particular coinductive data type. It will be interesting to consider a general form of coinductive data type in the spirit of Coq and the Calculus of Inductive Constructions [20–22, 53]. This will allow us to capture other notions of infinite data apart from streams such as infinite trees or equality between infinite objects .

As mentioned in Remark 3.10, in order to write primitive recursive functions we need to distinguish between data and coinductive data types and have two different fixed point operators. Recursive definitions on inductive data types in Coq are typed using another guardedness condition [22]. Roughly speaking, this condition constrains the arguments of a recursive call to be the pattern variables. It will be interesting to substitute this guardness condition for a typing rule as well.

The metrics considered for infinitary rewriting are smaller than ours [28, 29, 32]. It will be interesting to study infinitary extensions of term rewriting and combinatory reduction systems with different metrics. This will help us on the study of confluence and normalization for a calculus with a general form of coinductive data type.

Since type checking and type inference for pure type systems is decidable provided the system is weakly normalizing [50] we should be able to prove decidability for normalizing CoPTS's.

A variant of pure type systems, called *domain-free pure type systems*, λ-abstractions $\lambda x.b$ do not have annotated type is studied in [4]. It will be interesting to study the extension with corecursion of the domain-free pure type systems. This has the advantage of being closer to programming languages such as Haskell. Though, it also has disadvantages because we loose decidability of type inference when we move to second order polymorphic lambda calculus [52]. For this, we should also consider other variants of systems where second order quantifications are restricted as in [14, 43].

We have not added η-reduction because, as it is well-known, confluence of $\beta\eta$ on untypable terms with annotated types does not hold. The counterexample due to Nederpelt is $\lambda x{:}A.(\lambda y{:}B.y)x$ for $A \neq B$ [45]. A general confluence proof for weakly $\beta\eta$-normalizing PTS's is proved in [17]. It should be possible to adapt this proof to CoPTS's.

Acknowledgments

We would like to acknowledge Alexander Kurz, Tadeusz Litak and Daniela Petrişan for discussing the papers by Krishnaswami and Benton with us. We would also like to thank Neelakantan Krishnaswami for a helpful email exchange.

References

[1] S. Abramsky and C.-H. L. Ong. Full abstraction in the lazy lambda calculus. *Inform. and Comput.*, 105(2):159–267, 1993. ISSN 0890-5401.

[2] H. Barendregt. Lambda calculi with types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 118–310. Oxford University Press, 1992.

[3] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, Amsterdam, Revised edition, 1984. ISBN 0-444-86748-1; 0-444-87508-5.

[4] G. Barthe and M. H. Sørensen. Domain-free pure type systems. *J. Funct. Program.*, 10(5):417–452, 2000.

[5] S. Berardi. *Type Dependency and Constructive Mathematics*. PhD thesis, Carnegie Mellon University and Universitá di Torino, 1990.

[6] A. Berarducci. Infinite λ-calculus and non-sensible models. In *Logic and algebra (Pontignano, 1994)*, pages 339–377. Dekker, New York, 1996.

[7] L. Birkedal, J. Schwinghammer, and K. Støvring. A metric model of lambda calculus with guarded recursion. Presented at FICS 2010, 2010.

[8] L. Birkedal, R. E. Møgelberg, J. Schwinghammer, and K. Støvring. First steps in synthetic guarded domain theory: Step-indexing in the topos of trees. In *LICS*, pages 55–64, 2011.

[9] T. Borghuis. Modal pure type systems. *Journal of Logic, Language and Information*, 7(3):265–296, 1998.

[10] T. Coquand. Infinite objects in type theory. In *TYPES*, pages 62–78, 1993.

[11] T. Coquand and H. Herbelin. A - translation and looping combinators in pure type systems. *J. Funct. Program.*, 4(1):77–88, 1994.

[12] T. Coquand and G. P. Huet. The calculus of constructions. *Inf. Comput.*, 76(2/3):95–120, 1988.

[13] H. B. Curry and R. Feys. *Combinatory Logic*, volume I. North Holland, 1958.

[14] L. Damas and R. Milner. Principal type-schemes for functional programs. In *POPL*, pages 207–212, 1982.

[15] N. G. de Bruijn. A survey of the AUTOMATH project. In J. R. Hindley and J. Seldin, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980.

[16] J. Endrullis, C. Grabmayer, D. Hendriks, A. Isihara, and J. W. Klop. Productivity of stream definitions. *Theor. Comput. Sci.*, 411(4-5):765–782, 2010.

[17] H. Geuvers. The Church-Rosser property for beta-eta-reduction in typed lambda-calculi. In *LICS*, pages 453–460, 1992.

[18] H. Geuvers and M.-J. Nederhof. Modular proof of strong normalization for the calculus of constructions. *J. Funct. Program.*, 1(2): 155–189, 1991.

[19] H. Geuvers and J. Verkoelen. On fixed points and looping combinators in type theory. note, 2009.

[20] E. Giménez. *A Calculus of Infinite constructions and its applications to the verification of communicating systems*. PhD thesis, Ecole Normale Supérieure de Lyon, 1996.

[21] E. Giménez. Structural recursive definitions in type theory. In *ICALP*, pages 397–408, 1998.

[22] E. Giménez and P. Casterán. A tutorial on [co-]inductive types in coq. Technical report, Inria, 1998.

[23] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'aritmétique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.

[24] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *LICS*, pages 192–204, 1987.

[25] W. A. Howard. The formulae-as-types notion of construction. In J. R. Hindley and J. Seldin, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980.

[26] G. Hutton and M. Jaskelioff. Representing Contractive Functions on Streams. Submitted to the Journal of Functional Programming, 2011.

[27] R. Kennaway, J. W. Klop, M. R. Sleep, and F.-J. de Vries. Infinitary lambda calculi and Böhm models. In *RTA*, pages 257–270, 1995.

[28] R. Kennaway, J. W. Klop, M. R. Sleep, and F.-J. de Vries. Transfinite reductions in orthogonal term rewriting systems. *Inf. Comput.*, 119(1): 18–38, 1995.

[29] R. Kennaway, J. W. Klop, M. R. Sleep, and F.-J. de Vries. Infinitary lambda calculus. *Theor. Comput. Sci.*, 175(1):93–125, 1997.

[32] J. Ketema and J. G. Simonsen. Infinitary combinatory reduction systems. *Inf. Comput.*, 209(6):893–926, 2011.

[33] J. W. Klop. *Combinatory Reduction Systems*. PhD thesis, Rijkuniversiteit Utrecht, 1980.

[35] J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theor. Comput. Sci.*, 121 (1&2):279–308, 1993.

[36] N. R. Krishnaswami and N. Benton. Ultrametric semantics of reactive programs. In *LICS*, pages 257–266, 2011.

[37] N. R. Krishnaswami and N. Benton. A semantic model for graphical user interfaces. In *ICFP*, pages 45–57, 2011.

[38] N. R. Krishnaswami, N. Benton, and J. Hoffmann. Higher-order functional reactive programming in bounded space. In *POPL*, pages 45–58, 2012.

[39] J.-J. Lévy. An algebraic interpretation of the $\lambda\beta K$-calculus, and an application of a labelled λ-calculus. *Theoretical Computer Science*, 2 (1):97–114, 1976.

[40] G. Longo. Set-theoretical models of λ-calculus: theories, expansions, isomorphisms. *Ann. Pure Appl. Logic*, 24(2):153–188, 1983. ISSN 0168-0072.

[41] Z. Luo. ECC, an Extended Calculus of Constructions. In *Proceedings of LICS'89*, IEEE, pages 386–395. IEEE Computer Society Press, 1989.

[43] R. Milner. A theory of type polymorphism in programming. *J. Comput. Syst. Sci.*, 17(3):348–375, 1978.

[44] H. Nakano. A modality for recursion. In *LICS*, pages 255–266, 2000.

[45] R. P. Nederpelt. *Strong Normalization in a typed lambda calculus*. PhD thesis, Technische Universiteit Eindhoven, The Netherlands, 1973.

[46] J. C. Reynolds. Towards a theory of type structure. In *Symposium on Programming*, pages 408–423, 1974.

[48] P. Severi and E. Poll. Pure type systems with definitions. In *LFCS*, pages 316–328, 1994.

[49] J. Terlouw. Een nadere bewijstheoretische analyse van GSTT's. Manuscript, 1989.

[50] L. S. van Benthem Jutting. Typing in pure type systems. *Inf. Comput.*, 105(1):30–41, 1993.

[52] J. B. Wells. Typability and type-checking in the second-order lambda-calculus are equivalent and undecidable. In *LICS*, pages 176–185, 1994.

[53] B. Werner. *Une théorie des constructions inductives.* PhD thesis, Université Paris VII, 1994.

[54] H. Zantema and M. Raffelsieper. Proving productivity in infinite data structures. In *RTA*, pages 401–416, 2010.