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 normalisation. The set of finite and infinite terms is defined as a metric completion. We establish a precise connection between the modal operator (\(\bullet\)) and the metric at a syntactic level by relating a variable of type (\(\bullet\)\(A\)) with the depth of all its occurrences in a term. This syntactic connection between the modal operator and the depth 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:

\[
\text{zeros} = 0 : \text{zeros}
\]

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.

\[
\begin{align*}
\text{omegaprime} &= \text{tail} (0 : \text{omegaprime}) \\
\text{e} &= \text{filter} ((x \rightarrow (x>0)) \text{ zeros})
\end{align*}
\]

The last program does not produce the empty list, but it loops like the other two programs. Intuitively, the above programs are “badly behaved”. The idea of badly behaved programs is formalized in infinitary (weak) normalization 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.

\[
\begin{align*}
\text{omega} &= \text{omega} \\
\text{omegaprime} &= [\text{Integer}] \\
\text{e} &= [\text{Integer}]
\end{align*}
\]

The typed lambda calculus of reactive programs defined by Krishnaswami and Benton can type some recursive programs such as zerosprime and disallows terms like omega [33]. 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):

\[
\text{type DoubleFun a} = [a] \rightarrow [a] \rightarrow [a]
\]

3. Properties on streams and their proofs, using the Curry Howard isomorphism [13, 15, 25]. For example, we can have a constant

\[
\text{EqStr} : \Pi X : \text{set} . (\text{Stream} X) \rightarrow (\text{Stream} X) \rightarrow \text{prop}
\]

to represent equality between streams.

To give a more general presentation, we consider pure type systems (PTS’s) [2, 5, 44]. Pure type systems are a framework to define several existing typed lambda calculi à la Church in a uniform way. In particular, this includes the systems of the \(\lambda\) cube and the calculus of constructions [12]. We define Pure Type

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICFP 2012 Copenhagen, Denmark
Copyright © 2012 ACM [to be supplied]. . . $10.00

---

\(1\) 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.
Systems with Corecursion (CoPTS’s) by first extending the set of pseudoterms of a PTS with:

1. The type for streams (Stream $A$) with the constructor cons and the destructors head and tail.
2. The next modal type $(\bullet A)$, the constructor $\circ$ which moves one step after and a destructor await which moves one step before the moment in which the term is evaluated.
3. The fixed point operator to express corecursion which is denoted by $\text{cofix}$.

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 study the properties of infinitary weak and strong normalization for CoPTS’s. These notions are the analogues of weak and strong normalization. Proving infinitary weak normalization of a typed lambda calculus is a way of ensuring that all typable programs are well-behaved (it may not be the only way). Infinitary weak normalization gives at least one reduction strategy to the infinite normal form. Infinitary strong normalization expresses something stronger, namely, that any reduction strategy will find the normal form.

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 $C^\infty$ 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 $(\text{cons } a \ b)$ and $(\text{ob})$ is counted one deeper than the depth of the terms $(\text{cons } a \ b)$ and $(\text{ob})$ themselves. As a result, the set $C^\infty$ 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 [28–30].

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?

It is well-known that in lambda calculus the leftmost strategy is normalizing [3, Theorem 3.2.2]. However, when we consider infinite terms, the leftmost strategy is not longer infinitary normalizing. In the example,

$$\text{zeros} = \text{zeros}: \text{zeros}$$

the leftmost strategy does not lead to the infinite normal form in $\omega$-steps. We follow an infinitary normalizing strategy that reaches the normal form in $\omega$-steps which is a variation of the depth-first leftmost strategy. Figure 1 shows a tree representation of the infinite normal form of zeros which 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.

We establish a precise connection between the modal operator and the metric at a syntactic level by relating a variable of type $(\bullet A)$ with the depth of its occurrences in a term. This syntactic 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

---

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 zeros represented as a tree

```
(cons (cons 0 ·) ·)
```

Figure 2. The infinite normal form of zeros represented as a tree

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

Similarly,

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

For typing (cofix $x : A.b$), we require that $x_{i+1} : b : i A$. This means that the variable $x$ in (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 $\gamma$ be the reduction that unfolds fixed points:

$$(\text{cofix } x : A.b) \rightarrow, b[x := (\text{cofix } x : A.b)]$$

After contracting the fixed point, we have that the truncation of $b[x := (\text{cofix } x : A.b)]$ at depth 1 does not contain any residuals of the contracted redex. As an example, we consider the program zeros which is expressed in our syntax as follows.

$$\text{zeros} = (\text{cofix } x : (\text{Stream } \text{Nat}).(\text{cons } 0 \ x s))$$

The $\gamma$-redex occurs at depth 0 in zeros. We perform one $\gamma$-reduction step:

$$\text{zeros} \rightarrow, (\text{cons } 0 \text{ zeros})$$

In the future (after contracting the $\gamma$-redex), the $\gamma$-redex occurs at depth 1. The truncation of $(\text{cons } 0 \text{ zeros})$ at depth 1 which is $(\text{cons } 0 \bot)$ represents the information that has been displayed so far. The truncated subterm zeros which is at depth 1 in $(\text{cons } 0 \text{ zeros})$ 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 $C^\infty$ 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...
there is only one type constructor and only one reduction, namely 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 [38] and even inconsistent systems [23]. The word pure stands for the fact that there is only one type constructor and only one reduction, namely II 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 $S = (S, A, R)$ such that
1. $S$ is a set of symbols called sorts,
2. $A \subseteq S \times S$ called set of axioms,
3. $R \subseteq S \times S \times S$ 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 $\omega\nu$ (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 $A$ then $s_2 = s'_2$.
2. If $(s_1, s_2, s_3)$ and $(s_1, s_2, s'_3)$ are in $R$ then $s_3 = s'_3$.

Types and terms are defined in the same set $T$.

Definition 2.3 (Pseudoterm). The set $T_S$ (or $T$ for short) of pseudoterm is defined as follows.
$$T := \{ | S | (\lambda x:T).T | (T \cdot T) | (\Pi x:T).T \}$$

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

Definition 2.4 ($\beta$-Reduction). We define $\beta$-reduction as usual:
$$\beta \rightarrow \frac{(\lambda x:A.b)\ a \rightarrow b[x := a]}{(\beta)}$$

The relation $\rightarrow$ is defined as the smallest relations on pseudoterm which are closed under the $\beta$-rule and under contexts.

In the following section, we will define other notions of reduction such as $\sigma$ and $\pi$. We introduce the following notation which works for all of them.

Notation 2.5. Let $\rho$ be a notion of reduction.
1. $M \rightarrow_{\rho} N$ denotes a one step reduction from $M$ to $N$;
2. $M \rightarrow_{\rho} N$ denotes a finite reduction from $M$ to $N$;
3. $M \rightarrow_{\rho}^* N$ denotes a finite reduction from $M$ to $N$ with at least one step;
4. $M =_{\rho} N$ 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 pseudoterm for all $1 \leq i \leq n$.

Definition 2.6 (Pure Type System). A Pure Type System (PTS) denoted by $\lambda(S)$ is given by the judgement $\Gamma \vdash \alpha : A$ (or just $\Gamma \vdash \alpha : A$) 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 $\lambda$-cube). The systems of the $\lambda$-cube are obtained from the following set of sorts and axioms [2].
$$S = \{\text{type, kind}\} \quad A = \{\{\text{type}, \text{kind}\}\}$$

\begin{align*}
\text{(axiom)} & \vdash s_1 : s_2 \text{ if } (s_1, s_2) \in A \\
\text{(start)} & \Gamma \vdash A : s \\
\text{(weak)} & \Gamma \vdash \alpha : s \quad \Gamma \vdash \beta : B \\
\text{(prod)} & \Gamma \vdash (\Pi x:A.B) : s_3 \\
\text{(abs)} & \Gamma \vdash x:A \vdash B : s_3 \\
\text{(app)} & \Gamma \vdash (b : A) : B : s_3 \\
\text{(\beta-conv)} & \Gamma \vdash a : A \Gamma \vdash \alpha' : s \\
\end{align*}

Figure 2. Pure Type Systems

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

- (type, type) for terms depending on terms (functions),
- (kind, type) for terms depending on types (polymorphic functions),
- (type, kind) for types depending on terms (dependent types),
- (kind, kind) for types depending on type (type constructors).

The systems of the $\lambda$-cube consist of eight type systems. They all contain the rule (type, 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 $\lambda_{\rightarrow}$ is obtained from the specification $S_{\rightarrow}$ defined by the common sets $S$ and $A$ given above for the systems of the $\lambda$-cube and the following set of rules:

$$R = \{\{\text{type, type}\}\}$$

The second order lambda calculus $\lambda_{\rightarrow}$ is the pure type system $\lambda_2$ obtained from the following set of rules:

$$R = \{\{\text{type, type}, \{\text{kind, type}\}\}\}$$

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

$$R = \{\{\text{type, type}, \{\text{kind, type}\}, \{\text{kind, kind}\}\}\}$$

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

$$R = \{\{\text{type, type}, \{\text{kind, type}\}, \{\text{kind, kind}\}\}\}$$

Example 2.9 (Inconsistent Pure Type Systems). The system $\lambda V$ is given by the following specification (called $\lambda x$ in [2]).

$$S = \{\text{type}\} \quad A = \{\{\text{type, kind}\}\} \quad R = \{\{\text{type, type}\}\}$$

This system is inconsistent in the sense that all types are inhabited [2, 23]. For example, the circularity type $\mathbf{type} := \mathbf{type}$ is not necessary to derive inconsistency, see [2, Example 5.2.4]. In any inconsistent logical pure type system, a loop constructor can be derived from any term of type $\bot = \Pi x : X. X$ [11]. The paper [19] shows that Curry’s and Turing’s fixed point combinators $Y = \lambda f : (\lambda x . f (x x)) \rightarrow (\lambda x . f (x x))$ and $\Theta = (\lambda x . f (x x)) (\lambda x . f (x x))$ cannot be typed in $\lambda V$. 
Definition 2.10 (Term and Context). Let $S$ be a specification.

1. A (typable) term is a pseudoterm $a$ such that $\Gamma \vdash a : A$ for some $\Gamma$ and $A$.
2. A (legal) context is a pseudocOntext $\Gamma$ such that $\Gamma \vdash a : A$ for some $a$ and $A$.

In the following definition, we consider an arbitrary reduction $\rho$. In later sections, we will define other notions of reductions besides $\beta$.

Definition 2.11 (Weak and Strong Normalization). Let $\rho$ be a notion of reduction.

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

Definition 2.12 (Weakly and Strongly Normalizing PTS). We say that $\lambda(S)$ is strongly (weakly) $\beta$-normalizing if for all $\Gamma \vdash a : A$ we have that $a$ and $A$ are strongly (weakly) $\beta$-normalizing.

Notation 2.13. \[\lambda(S) \models \rho\text{-WN} \iff \lambda(S) \text{ is weakly } \rho\text{-normalizing.}\]

\[\lambda(S) \models \rho\text{-SN} \iff \lambda(S) \text{ is strongly } \rho\text{-normalizing.}\]

Obviously, $\lambda(S) \models \rho\text{-SN}$ implies $\lambda(S) \models \rho\text{-WN}$. A proof of the following result can be found in [2].

Theorem 2.14 (Strong Normalization of $\lambda(C)$). We have that $\lambda(C) \models \beta\text{-SN}$.

The following result is proved in [2, Proposition 5.2.31]. We use the abbreviation $\Downarrow = \Pi X :: X$.

Theorem 2.15 (Inconsistent implies not normalizing). Let $\lambda(S)$ be a PTS extending $\lambda X$. Suppose $\Gamma \vdash a : A$. Then, $a$ is not weakly $\beta$-normalizing. Hence, $\lambda(S) \not\models \beta\text{-WN}$.

As a consequence of the previous theorem, the inconsistent pure type system $\lambda V$ from Example 2.9 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 $\text{(Stream } A\text{)}$ for streams of type $A$, $(\bullet A)$ for the modality next and a fixed point operator $\text{(cofix } x : A.a\text{)}$ for expressing corecursion.

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

\[
\mathcal{C} ::= V | S | (\lambda V : C.C) | C.C | (\Pi V : C.C) \\
\bullet C | o C | (\text{await } C) \\
(\text{Stream } C) | (\text{cons } C.C) | (\text{hd } C) | (\text{tl } C) \\
(\text{cofix } V : C.C)
\]

We introduce two notions of reductions apart from $\beta, \sigma$ for computing the head and tail of a stream and $\gamma$ for unfolding fixed points.

Definition 3.2 ($\sigma$ and $\gamma$-Reductions). We define the following reduction rules:

\[
\begin{align*}
(\text{await } (oa)) & \rightarrow a \\
(\text{hd } (\text{cons } a b)) & \rightarrow a \\
(\text{tl } (\text{cons } a b)) & \rightarrow b \\
(\text{cofix } x : A.b) & \rightarrow b[x := (\text{cofix } x : A.b)]
\end{align*}
\]

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

(axiom) $\Gamma \vdash s_1 : s_2$ if $(s_1, s_2) \in A$

(start) $\Gamma \vdash A_i : s$, $j \geq i \implies x_i : A \overset{\Gamma\text{-fresh}}{\rightarrow} x_i : A$

(weak) $\Gamma \vdash A_i : s$, $\Gamma \vdash b_i : B \implies x_i : A \overset{\Gamma\text{-fresh}}{\rightarrow} b_i : B$

(prod) $\Gamma, x_i : A \vdash b_i : B \implies \Gamma \vdash (\Pi x : A.B) : s \overset{\Gamma\text{-fresh}}{\rightarrow} \Gamma \vdash (\lambda x : A.B) : s$

(abs) $\Gamma, x_i : A \vdash b_i : B \implies \Gamma \vdash \lambda x : A.B : s \overset{\Gamma\text{-fresh}}{\rightarrow} \Gamma \vdash (\lambda x : A.B) : s$

(app) $\Gamma \vdash b_i : (\Pi x : A.B) \implies \Gamma \vdash a_i : A \implies \Gamma \vdash (b_i : (\Pi x : A.B)) : s \overset{\Gamma\text{-fresh}}{\rightarrow} \Gamma \vdash (\lambda x : A.B) : s$

($\beta\sigma\gamma$-conv) $\Gamma \vdash a_i : A \implies \Gamma \vdash A' : s \implies \Gamma \vdash a_i : A \overset{\Gamma\text{-fresh}}{\rightarrow} A =_{\beta\sigma\gamma} A'$

(mod) $\Gamma \vdash A_i : \text{type} \implies \Gamma \vdash \tilde{a}_i : A$

($\bullet I$) $\Gamma \vdash a_i : A \implies \Gamma \vdash (\text{await } a_i) : s$

($\bullet E$) $\Gamma \vdash (\text{await } a_i) : s \implies \Gamma \vdash a_i : A$

(stream) $\Gamma \vdash A_i : \text{type} \implies \Gamma \vdash (\text{Stream } A) : s$

(cons) $\Gamma \vdash a_i : A \implies \Gamma \vdash b_i : B \implies \Gamma \vdash (\text{cons } a_i b) : (\text{Stream } A) \overset{\Gamma\text{-fresh}}{\rightarrow} \Gamma \vdash (\text{cons } a_i b) : (\text{Stream } A)$

(hd) $\Gamma \vdash a_i : (\text{Stream } A) \implies \Gamma \vdash (\text{hd } a_i) : A$

(tl) $\Gamma \vdash a_i : (\text{Stream } A) \implies \Gamma \vdash (\text{tl } a_i) : (\text{Stream } A)$

(cofix) $\Gamma \vdash \text{cofix } x : A.b : s \implies \Gamma \vdash \text{cofix } x : A.b : s$

}\]

Figure 3. Pure Type Systems with Corecursion on Streams

Judgements of CoPTS’s are of the form $\Gamma \vdash a : A$ where $i$ is an index representing “time”. A pseudocOntext $\Gamma = x_i : A_1, x_{i+1} : A_2, \ldots, x_n : A_n$ 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 $\mathcal{S}_a$ is the specification for the simply typed lambda calculus defined in Example 2.8.

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

Example 3.4 (Typed $\lambda$-calculus of Reactive Programs as a CoPTS). Krishnaswami and Benton’s typed lambda calculus presented in
[33] 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 $\lambda^\omega$. 

**Remark 3.5 (Alternative Typing Rules for cofix using Modality).** As in [33, 34], 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 [34]. 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.

$$
\begin{array}{c}
\text{cofix} \\
\frac{\Gamma, x : \bullet A \vdash b_i A}{\Gamma \vdash A \vdash \text{cofix} \cdot x : \bullet A, b_i A}
\end{array}
$$

The typing rules (cofix) and (cofix') are equivalent. The rule (cofix) allows us to derive (cofix') by defining $\text{cofix}' x : A. b = \text{cofix} y : A. b [x := (\text{cofix}' y : A. b)]$. Conversely, we can set $\text{cofix} y : A. b = \text{cofix}' x : A. b [y := (\text{cofix}' \cdot A \to A) \to A]$, for all $i$ as in [33].

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 system using the rule (cofix'), our programmes would have been burdened with modalities. For example, let’s write the example of zeros given in the introduction using (cofix').

$$
\text{zeros}' = (\text{cofix}' x : \bullet (\text{Stream Nat}). (\text{cons} 0 (\text{await} x)))
$$

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 [33] (examples where modalities are necessary can be found in [33–35]). Nakano’s type system has modalities without indices with help of subtyping and recursive types [40]. 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 give examples of terms typable in CoPTS’s. We define a context $\Gamma_{\text{Nat}}$ containing the following type declarations:

$$\begin{align*}
\text{Nat} & : \text{type} \\
0 & : \text{Nat} \\
\text{suc} & : \text{Nat} \to \text{Nat} \\
+ & : \text{Nat} \to \text{Nat} \to \text{Nat} \\
* & : \text{Nat} \to \text{Nat} \\
\text{Bool} & : \text{type} \\
< & : \text{Nat} \to \text{Nat} \to \text{Bool} \\
\text{if} & : \text{Bool} \to (\text{Stream Nat}) \to (\text{Stream Nat})
\end{align*}$$

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 $\lambda^\omega$).** Define the following:

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{FunSNat}$</td>
<td>$(\text{Stream Nat}) \to (\text{Stream Nat}) \to (\text{Stream Nat})$</td>
</tr>
<tr>
<td>$\text{zeros}$</td>
<td>$(\text{cofix} \cdot x : (\text{Stream Nat}). (\text{cons} 0 x))$</td>
</tr>
<tr>
<td>$\text{interleave}$</td>
<td>$\text{cofix} f : \text{FunSNat}$. $\lambda x : (\text{Stream Nat})$. $\lambda y : (\text{Stream Nat})$. $\text{cons} \left( (\text{hd} x) \left( \text{fun} \left( f y \left( (\text{tl} x) y \right) \right) \right) \right)$</td>
</tr>
<tr>
<td>$\text{sumlist}$</td>
<td>$\text{cofix} f : \text{FunSNat}$. $\lambda x : (\text{Stream Nat})$. $\lambda y : (\text{Stream Nat})$. $\text{map} \left( (\text{cons} \left( x \right) y) \right)$</td>
</tr>
<tr>
<td>$\text{merge}$</td>
<td>$\text{cofix} f : \text{FunSNat}$. $\lambda x : (\text{Stream Nat})$. $\lambda y : (\text{Stream Nat})$. $\text{if} (x) (y)$</td>
</tr>
</tbody>
</table>

We have that all the above terms can be typed in $\lambda^\omega$.

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma_{\text{Nat}}$</td>
<td>$\vdash \text{zeros} : (\text{Stream Nat})$</td>
</tr>
<tr>
<td>$\Gamma_{\text{Nat}}$</td>
<td>$\vdash \text{interleave} : : \text{FunSNat}$</td>
</tr>
<tr>
<td>$\Gamma_{\text{Nat}}$</td>
<td>$\vdash \text{sumlist} : : \text{FunSNat}$</td>
</tr>
<tr>
<td>$\Gamma_{\text{Nat}}$</td>
<td>$\vdash \text{merge} : : \text{FunSNat}$</td>
</tr>
</tbody>
</table>

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

$$
\begin{align*}
\text{map} & \equiv \lambda X : \text{type}. \\
 & \quad \lambda Y : \text{type}. \\
 & \quad \lambda g : X \to Y. \\
 & \quad (\text{cofix} f : (\text{Stream } X) \to (\text{Stream } Y). \\
 & \quad (\text{cons} \left( g (\text{hd} x) \right) (\text{fun} \left( f (\text{tl} x) y \right) \right) ))
\end{align*}
$$

can be typed in $\lambda^\omega$, i.e.

$$
\begin{align*}
\vdash \text{map} : & \Pi X : \text{type}. \Pi Y : \text{type}. \\
 & (X \to Y) \to (\text{Stream } X) \to (\text{Stream } Y)
\end{align*}
$$

We can also write type constructors such as:

$$
\begin{align*}
\text{DoubleFun} & \equiv \lambda X : \text{type}. \\
 & \quad (\text{Stream } X) \to (\text{Stream } X) \to (\text{Stream } X)
\end{align*}
$$

which can be typed in $\lambda^\omega$ as follows.

$$
\vdash \text{DoubleFun} : : \text{type} \to \text{type}
$$

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

$$
\begin{align*}
\Gamma_{\text{Nat}}, \text{EqStr} : & \Pi X : \text{type}. (\text{Stream } X) \to (\text{Stream } X) \to \text{type} \\
\vdash & \text{EqStr } \text{Nat } \text{zeros } \text{zeros}' : \text{type}
\end{align*}
$$

**Example 3.8 (Typable Terms in CoPTS not satisfying guardedness condition).** 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 can all be typed in $\lambda^\omega 2$ but they do not satisfy the guardedness condition. Let $\text{map} := \text{map Nat}$. 
\[
\text{zeros}' = (\text{cofix } x.:(\text{Stream Nat}.
\quad \text{cons } 0 \text{ (interleave } x \times x))
\]
\[
\text{fib} = (\text{cofix } x.:(\text{Stream Nat}.
\quad \text{cons } 1 \text{ (cons } 1 \text{ (sumlist } x \times (\text{tl } x)))
\]
\[
\text{hamming} = (\text{cofix } h.:(\text{Stream Nat}.
\quad \text{cons } 1
\quad \text{merge}
\quad (\text{map } (\lambda x.:\text{Nat}.2 \times x) h)
\quad \text{merge}
\quad (\text{map}(\lambda x.:\text{Nat}.3 \times x) h)
\quad (\text{mapn}(\lambda x.:\text{Nat}.5 \times x) h))
\]
They can all be typed in $\lambda^\omega 2$ as follows.
\[
\Gamma_{\text{Nat}} \vdash \text{zeros}' :: \text{Nat}
\Gamma_{\text{Nat}} \vdash \text{fib} :: \text{Nat}
\Gamma_{\text{Nat}} \vdash \text{hamming} :: \text{Nat}
\]
We formalize the badly behaved Haskell programmes given in the introduction in our setting and show that they are not typable.

**Example 3.9 (The Undesirables).** The badly behaved programmes shown in the introduction can be written in our syntax as follows.
\[
\Omega = (\text{cofix } x.:(\text{Stream X}.x))
\]
\[
\Omega_{\text{tail}} = (\text{cofix } x.:(\text{Stream X}.) (\text{tl } x))
\]
\[
\Omega' = (\text{cofix } x.:(\text{Stream X}.)(\text{cons } 0 x))
\]
\[
\Omega'' = (\text{cofix } x.:(\text{Stream X}.)(\text{consn } x))
\]
\[
E = \text{filter Nat}(\lambda x.:(\text{Stream Nat}.) x > 0 \text{ zeros})
\]
where the function filter is defined as follows:
\[
\text{filter} = \lambda X : \text{type} \lambda \mathcal{P} : X \rightarrow \text{Bool}
\quad \text{cofix } f.:(\text{Stream X}.) \rightarrow (\text{Stream X}).
\quad \lambda \mathcal{X} .:(\text{Stream X}.)
\quad \text{if } (P \text{ hd } x) \text{ then }
\quad \text{(cons } \text{ hd } x \text{ } f \text{ (tl } x))
\quad \text{else }
\quad \text{(tl } x)
\]
None of the above terms are typable in any CoPTS. More formally, we have that the following holds for all $A$ and $i$:
\[
A :: \text{type } \Gamma :: A
\]
\[
A :: \text{type } \Omega :: A
\]
\[
A :: \text{type } \Omega_{\text{tail}} :: A
\]
\[
\Gamma_{\text{Nat}} :: \text{filter} :: A
\]
\[
\Gamma_{\text{Nat}} :: E :: A
\]
The terms $\Omega$, $\Omega_{\text{tail}}$, and $\text{filter}$ 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 $\Omega'$ and $\Omega''$ are not typable because they would reduce $\Omega$ which is not typable (Theorem 4.5). The term $E$ is not typable because it has a subterm which is not typable.

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

**Definition 3.10 (Pure Type System with Corecursion from $n$).** Let $S$ be a specification extending $S_n$ and $n \in \mathbb{N}$. A Pure Type System with Corecursion on Streams from $n$ (CoPTS$n$) denoted by $\lambda^\omega n (S)$ is given by the judgement $\Gamma \vdash a :: i ; A$ (or just $\Gamma \vdash a :: A$) for $i \in \mathbb{N}$ and defined by replacing the rule (cofix) from the typing rules of Figure 3 by the following one:
\[
(\text{cofix})^n \quad \frac{\Gamma ; x^i ; A \vdash b :: A \quad \Gamma \vdash A :: A :: \text{type} \quad j \geq n}{\Gamma \vdash (\text{cofix } x.:(A:b),A) :: j}
\]

### 4. Basic Properties

In this section we prove some basic properties on CoPTS$n$’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).** $(C, \rightarrow_{\beta \sigma \gamma})$ is confluent.

**Proof.** This follows from [32, Corollary 13.6] (see also [31]) by observing that $(C, \rightarrow_{\beta \sigma \gamma})$ is an orthogonal combinatory reduction system.

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

**Proof.** Observe that the number of symbols decreases in each $\sigma$-reduction step.

The notation $\Gamma_{++}$ means that we add $k$ to the index of every hypothesis in $\Gamma$.

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

The above theorem is proved by induction on the derivation.

**Lemma 4.4 (Substitution).** If $\Gamma \vdash a :: A$ and $\Gamma, x :: A, \Gamma' \vdash b :: B$ then $\Gamma, \Gamma'[x := a] : A \vdash b :: B[x := a]$.

**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 $a \rightarrow_{\beta \sigma \gamma} a'$. If $\Gamma \vdash a :: A$ then $\Gamma \vdash a' :: A$.

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

1. If $\Gamma \vdash a :: A$ and $a \rightarrow_{\beta \sigma \gamma} a'$ then $\Gamma \vdash a' :: A$.
2. If $\Gamma \vdash a :: A$ and $\Gamma \rightarrow_{\beta \sigma \gamma} \Gamma'$ then $\Gamma' \vdash a :: A$.

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 $S$ be single sorted. If $\Gamma \vdash a :: A$ and $\Gamma \vdash a :: A'$ then $A \approx_{\beta \sigma \gamma} A'$.

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

**Definition 4.7 (Strongly Normalizing CoPTS).** Let $\rho$ be a notion of reduction. We say that $\lambda^\omega (S)$ is weakly (strongly) $\rho$-normalizing if for all $\Gamma \vdash a :: A$, we have that $A = A'$ where $A$ is weakly (strongly) $\rho$-normalizing.

**Notation 4.8.** $\lambda^\omega (S) \models \rho - \text{WN (SN)}$ if $\lambda^\omega (S)$ is weakly (strongly) $\rho$-normalizing.

We use the following abbreviations:
\[
\downarrow = \Pi X. : \text{type}, X
\]
\[
S = \lambda X. : \Pi Y. : \text{type}, (X \rightarrow Y \rightarrow Y) \rightarrow Y
\]
We consider the context $\Gamma_0$ defined as $c : \downarrow$ where $c$ is ‘fresh’.
Definition 4.9 (Encoding in $\lambda \omega$). Let $\Gamma \vdash d : D$. We define $\{d\}$ by induction on $d$.

\[
\begin{align*}
\{x\} &= x \\
\{s\} &= s \\
\{\Pi x : A.B\} &= \Pi \{x\} : \{A\}. \{B\} \\
\{\lambda x : A.b\} &= \lambda \{x\} : \{A\}. \{b\} \\
\{(a \ b)\} &= \{(a\} \ (\{b\}) \\
\{a.\} &= \{a\} \\
\{\{\text{await} \ a\} \} &= \{a\} \\
\{(\text{Stream} \ A)\} &= S \{A\} \\
\{(\text{cons} \ a \ b)\} &= \lambda Y : \text{type}. \lambda f : A_0 \rightarrow Y \rightarrow Y. f \{a\} \ (\{b\} \ Y \ f) \\
\{(\text{hd} \ a)\} &= \{a\} \ A_0 (\lambda x : A_0. \lambda y : A_0 \cdot x) \\
\{(\text{tl} \ a)\} &= \{a\} \ (S \ A_0) (\lambda x : (S \ A_0). \lambda y : (S \ A_0) \cdot y) \\
\{(\text{cofix} \ x : A.b)\} &= (\lambda x : \{A\}. \{b\}) \ (\{A\})
\end{align*}
\]

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

$\Gamma \vdash (\text{cons} \ a \ b) : (\text{Stream} \ A)$

$\Gamma \vdash (\text{tl} \ a) : (\text{Stream} \ A)$

$\Gamma \vdash (\text{hd} \ a) : A$

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

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

The following statements are not difficult to prove.

Theorem 4.10. 1. If $a \rightarrow_\sigma a'$ then $\{a\} \rightarrow_\beta \{a'\}$.

2. If $a \rightarrow_\rho a'$ then $\{a\} \rightarrow_\beta \{a'\}$.

Theorem 4.11 (Encoding from $\lambda^{\omega} \omega$ to $\lambda \omega$). If $\Gamma \vdash d : D$ then $(\Gamma), \{d\}, \{D\}$ are well defined and $\Gamma_0, (\Gamma) \vdash \{d\} : \{D\}$.

Proof. This follows by induction on the structure of the term using Generation Lemma. We show the case $d = (\text{cons} \ a \ b)$. Suppose $\Gamma \vdash (\text{cons} \ a \ b) : (\text{Stream} \ A)$ and $\Gamma \vdash (\text{cons} \ a \ b) : i (\text{Stream} \ A')$. Note that in $\lambda^{\omega} \omega$, we only have $\beta$-conversion without $\sigma \gamma$. It follows from Theorem 4.6 that $A = \beta A'$. By Theorem 4.10, we have that $\{A\} = 0 \{A'\}$, Hence, $\Gamma \vdash a : A$ and $\Gamma \vdash a : A'$. By Induction Hypothesis, $\Gamma \vdash \{A\} : \{A\}$ and $\Gamma \vdash \{a\} : \{A'\}$. Since $\lambda \omega$ is strongly $\beta$-normalizing, $A_0$ from Definition 4.9 is uniquely determined since the $\beta$-normal forms of $A$ and $A'$ are the same. Hence, $(\Gamma) \vdash d$ is well defined.

Theorem 4.12 (Strong Normalization of $\lambda^{\omega} \omega$ without Contracting Fixpoints). $\lambda^{\omega} \omega \vdash \beta \gamma$-SN.

Proof. Suppose $\Gamma \vdash a : A$. By Theorem 4.11, we have that $\{a\}$ is typable in $\lambda \omega$ and hence, it is $\beta \gamma$-normalizing. We prove that $a$ is strongly $\beta \gamma$-normalizing by contradiction. Suppose that $a$ is not strongly $\beta \gamma$-normalizing. That is, suppose there exists an infinite $\beta \gamma$-reduction sequence starting from $a$. Observe that the number of $\beta$-reduction steps in this sequence must be infinite because $\sigma$ is strongly normalizing (Theorem 4.2). Hence, the sequence is of the form:

$\begin{align*}
a &= a_0 \rightarrow_\sigma a_1 \rightarrow_\beta a_2 \rightarrow_\sigma a_3 \rightarrow_\beta a_4 \rightarrow_\sigma a_5 \rightarrow_\beta a_6 \ldots
\end{align*}$

By Theorem 4.10, we have that:

$\begin{align*}
\{a\} &= \{a_0\} \rightarrow_\beta \{a_1\} \rightarrow_\beta \{a_2\} \rightarrow_\beta \{a_3\} \rightarrow_\beta \{a_4\} \rightarrow_\beta \{a_5\} \rightarrow_\beta \{a_6\} \ldots
\end{align*}$

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

In order to prove that $\lambda \omega \vdash \beta \gamma$-SN implies $\lambda C \vdash \beta \gamma$-SN given in [2] to CoPTS’s.

Definition 4.13. We consider $\lambda^{\omega} (C)$.

- We say that $A$ is a kind if $\Gamma \vdash \omega A : \text{kind for some} \ \Gamma$.
- We say that $A$ is a type constructor if $\Gamma \vdash C : A : B : \text{kind for some} \ \Gamma$ and $B$.
- We say that $a$ is an object if $\Gamma \vdash a : A : \text{type for some} \ \Gamma$ and $A$.

We consider the context $\Gamma_1$ defined as $0$-type, $c.\bot$, where $0, c$ are ‘fresh’. As in [2], we define three mappings:

1. The mapping $\rho$ on kinds is exactly as in [2, Definition 5.3.3].

2. The mapping $\tau$ on type constructors and kinds is the extension of [2, Definition 5.3.7] with the following clauses:

$\tau(a) = \bullet \tau(a)$

$\tau(\text{Stream} \ A) = \tau(\text{Stream} \ A)$

3. The mapping $[\cdot]$ on objects, type constructors and kinds is the extension of [2, Definition 5.10] with the following clauses:

$[\bullet A] = c (0 \rightarrow 0) [A]$ $[\text{cons} \ a \ b] = [\{ \text{cons} \ a \ b \} \ A_0]$ $[\{\text{await} \ a\} ] = [\text{await} [a]]$ $[\{\text{Stream} \ A\} ] = c (0 \rightarrow 0) [A]$ $[\{\text{cons} \ a \ b\} ] = [\{ \text{cons} \ a \ b \} \ b]$ $[\{\text{hd} \ a\} ] = [\text{hd} [a]]$ $[\{\text{tl} \ a\} ] = [\text{tl} [a]]$ $[\{\text{cofix} \ x : A.b\} ] = \lambda (\lambda (\cdot : (\{A\}. \{b\})) \ (\{A\}))$ $[\text{cofix} x : A.b] = \lambda (\cdot : (\lambda (\cdot : (\{A\}. \{b\})) \ (\{A\})))$

Lemma 4.14 (Mapping on kinds). Let $\Gamma \vdash C : A : \text{kind}$. Let $\Gamma \vdash C : A : \text{kind}$.

1. $\Gamma \vdash \omega \rho (A) : \text{kind}.$

2. If $A \rightarrow_\beta A'$ then $\rho (A) \equiv \rho (A')$.

The first statement follows by induction on the derivation. The second one follows by induction on the structure of $A$.

Lemma 4.15 (Mapping on type constructors and kinds). Let $\Gamma \vdash C : A : B$ where $\Gamma \vdash C : A : B : \text{kind or} B : \text{kind}$. Let $\Gamma \vdash C : A : B$ where $\Gamma \vdash C : A : B : \text{kind or} B : \text{kind}$.

1. $\Gamma \vdash \omega \tau (A) : \tau (B)$. $\Gamma \vdash \omega \tau (A) : \tau (B)$.

2. If $A \rightarrow_\beta A'$ then $\tau (A) \rightarrow_\beta \tau (A')$.

The first statement follows by induction on the derivation using Lemma 4.14. The second one follows by induction on the structure of $A$ observing that $\tau$ deletes the objects which are the only ones that can contain $\sigma \gamma$-redexes.

Lemma 4.16 (Mapping on objects, type constructors and kinds). Let $\Gamma \vdash C : A : \text{kind}.

1. $\tau (\Gamma) \vdash \omega \tau (a) : \tau (A)$.

2. If $a \rightarrow_\beta a'$ then $[\cdot] [a] \rightarrow_\beta [\cdot] [a']$.

The first statement follows by induction on the derivation using Lemma 4.15. The second one follows by induction on the structure of $a$.

Theorem 4.17. $\lambda^{\omega} (C) \vdash \beta \gamma$-SN.

Proof. Suppose $a$ is typable in $\lambda^{\omega} (C)$. By Lemma 4.16 part (1), $[\cdot] [a]$ is typable in $\lambda^{\omega} \omega$. Suppose towards a contradiction that there exists an infinite $\beta \gamma$-reduction sequence starting from $a$. By Lemma 4.16 part (2), there also exists an infinite $\beta \sigma$-reduction starting from $[\cdot] [a]$. This contradicts Theorem 4.12.

Remark 4.18. The previous theorem is about $\beta \sigma$-reduction and does not mention $\gamma$ for the reason, that CoPTS’s can not be $\gamma$-normalizing, as terms containing a fixed point may have an infinite $\gamma$-reduction.
A : | f ; : •A → A ⊢ (cofix x:A. f (f x)) : A

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{align*}
   a|_p &= a \\
   \Pi x:A. B|_p &= A|_p \\
   \lambda x:A. b|_p &= A|_p \\
   (a b)|_p &= a|_p \\
   (\bullet A)|_p &= a|_p \\
   (\text{await} a)|_p &= a|_p \\
   (\text{Stream} A)|_p &= A|_p \\
   (\text{cons} a b)|_p &= a|_p \\
   (\text{hd} a)|_p &= a|_p \\
   (\text{tl} a)|_p &= a|_p \\
   \text{cofix} x:A.b|_p &= A|_p
\end{align*}

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

Let p, q be two positions, i.e., sequences of 0’s and 1’s. We define p < q if there exists a non-empty position r such that q = r.p.

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 (cons c d) or (oc).

For example, the depth of (cons 3 (cons 5 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 ⊥.

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 ⊥. Equivalently, it can be defined by induction as follows.

\begin{align*}
   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} \\
   (a b)^{n+1} &= (a^{n+1} | b^{n+1}) \\
   (\bullet A)^{n+1} &= \bullet A^{n+1} \\
   (\text{cofix} x:A.b)^{n+1} &= (\text{cofix} x:A^{n+1}. b^{n+1})
\end{align*}

Example 5.7 (What is not in C∞?). The following “terms” do not belong to C∞.

\begin{align*}
   (f (f \ldots)) \\
   \lambda x_1. \lambda x_2. \lambda x_3. \ldots \\
   (((\ldots x_2). x_2). x_1) \\
   (\text{tl} (\text{tl} (\text{tl} \ldots)))
\end{align*}

The first three terms are characteristic examples of respectively a Böhm tree, Lévy Longo and Berarducci tree [1, 3, 6, 28, 36, 37]. 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, 30].
Notation 5.8 (Reduction at depth \( n \)). We denote \( \rho \xrightarrow{\gamma} b \) if the contracted \( \rho \)-redex is at depth \( n \).

Definition 5.9 (Strongly Converging Reductions). A strongly convergent \( \rho \)-reduction sequence of length \( \alpha \) (an ordinal) is a sequence \( \{ a_\beta \mid \beta \leq \alpha \} \) of terms in \( C^\infty \), such that:
1. \( a_\beta \xrightarrow{\rho} a_{\beta+1} \) for all \( \beta < \alpha \).
2. \( a_\alpha = \lim_{\beta < \lambda} a_\beta \) for every limit ordinal \( \lambda \leq \alpha \).
3. \( \lim_{i \to \infty} d_i = 0 \) where \( d_i \) is the depth of the redex contracted at \( a_i \xrightarrow{\rho} a_{i+1} \) for every limit ordinal \( \lambda \leq \alpha \).

Notation 5.10 (Strongly convergent reduction). \( a \xrightarrow{\rho} b \) denotes a strongly converging reduction from \( a \) to \( b \).

By construction, the set \( C^\infty \) is closed under strongly converging reduction.

Example 5.11 (Strongly Converging Reductions). We have that
\[
\begin{align*}
zeros & \xrightarrow{} (\text{cons } 0 ) \\
\lim & \xrightarrow{} (\text{cons } 0 \text{ zeros) } \\
\lim & \xrightarrow{} (\text{cons } 0 \text{ (cons } 0 \text{ zeros) }) \\
\lim & \xrightarrow{} (\text{cons } 0 \text{ (cons } 0 \text{ (cons } 0 \text{ 0 } \ldots)) \\
\ldots & \\
\lim & \xrightarrow{} (\text{cons } 0 \text{ (cons } 0 \text{ (cons } 0 \text{ 0 } \ldots)) )
\end{align*}
\]
Let \( \text{nzeros} = (\text{cons } 0 \text{ (cons } 0 \text{ (cons } 0 \text{ 0 } \ldots)) ) \) be the infinite normal form of \( \text{nzeros} \) (see Definition 6.1). We show an example of a reduction sequence of length \( \omega \) (we indicate the depth of the redex in the superscript of the rewrite arrows):

\[
\begin{align*}
zeros & \xrightarrow{} (\text{cons } 0 \text{ zeros) } \\
zeros & \xrightarrow{} (\text{cons } 0 \text{ (cons } 0 \text{ zeros) }) \\
zeros & \xrightarrow{} (\text{cons } 0 \text{ (cons } 0 \text{ (cons } 0 \text{ 0 } \ldots)) \\
\ldots & \\
\lim & \xrightarrow{} (\text{cons } 0 \text{ (cons } 0 \text{ (cons } 0 \text{ 0 } \ldots)) )
\end{align*}
\]
As we mentioned in the introduction, there exists a strongly converging reduction sequence of length \( \omega \) from \( \text{nzeros} \) 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:
\[
\begin{align*}
\Omega & \xrightarrow{\gamma} \Omega \\
\Omega & \xrightarrow{\gamma} (\text{tl } \Omega) \\
\Omega & \xrightarrow{\gamma} (\text{tl } \Omega) \\
\text{(cofix } x:A,f x) & \xrightarrow{\gamma} \text{(cofix } x:A,f x)
\end{align*}
\]

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 it is weakly \( \beta\sigma \)-normalizing. Proving infinitary weak normalization poses several difficulties:
1. Contracting \( \gamma \)-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 \( \Gamma \upharpoonright n \).

Definition 6.1 (Infinitary Weak Normalization). Let \( \rho \) be a notion of reduction. We say that \( a \) is infinitary weakly \( \rho \)-normalizing if there exists a \( \rho \)-normal form \( b \) such that \( a \xrightarrow{\rho} b \). In this case, we say that \( b \) is the (infinitary) \( \rho \)-normal form of \( a \).

The undesirable terms (see Example 3.9) are not infinitary weakly \( \beta\sigma\gamma \)-normalizing. The term \( (\text{map Nat Nat } x) \) is weakly \( \beta\sigma\gamma \)-normalizing. Its normal form is depicted as a tree in Figure 5. Our tree representation reflects the notion of depth.

Definition 6.2 (Infinitary Weak Normalizing CoPTS). We say that \( \lambda^\infty(S) \) is infinitary weakly \( \rho \)-normalizing if for all \( a \in C \) such that \( \Gamma \vdash a : A \), we have that \( a \) is infinitary weakly \( \rho \)-normalizing.

Notation 6.3. \( \lambda^\infty(S) \vdash \rho \text{-WN} \) if \( \lambda^\infty(S) \) is infinitary weakly \( \rho \)-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 \( n \geq 1 \). If \( \Gamma \vdash a : A \) then \( a^{n-1} \) is in \( \gamma \)-normal form, i.e. \( a^{n-1} \) does not have fixed points.

Proof. We prove it simultaneously with the statement: if \( x ; B \) is in \( \Gamma \) then \( B^{n-1} \) is in \( \gamma \)-normal form.

We define a function that contracts all \( \text{cofix} \) occurrences of a pseudoterm just once.

Definition 6.5. We define \([ a ]\) by induction on \( a \).
\[
\begin{align*}
[x] & = x \\
[s] & = s \\
[\Pi x:A.b] & = \Pi x: [A]. [b] \\
[\lambda x:A.b] & = \lambda x: [A]. [b] \\
[(a b)] & = ([a] [b] ) \\
[\text{cofix } a] & = [a] \\
[\langle \text{stream } A \rangle] & = [\text{stream } A] \\
[(\text{cons } a b)] & = (\text{cons } [a] [b]) \\
[\langle \text{hd } a \rangle] & = [\text{hd } a] \\
[\langle \text{tl } a \rangle] & = [\text{tl } a] \\
[\langle \text{cofix } x:A.b \rangle] & = [b] [x := (\text{cofix } x: [A], [b]) ]
\end{align*}
\]

The map \([ ]\) is extended to contexts in the obvious way.
\[
[x_1 ; i_1 A_1, \ldots, x_n ; i_n A_n ] = x_1 ; i_1 [A_1], \ldots, x_n ; i_n [A_n]
\]

Note that \( a \xrightarrow{\gamma} [a]\).

Theorem 6.6. Let \( \Gamma \vdash a : A \). Then \( \Gamma \vdash a^{n+1} [a]; i [A] \).

Proof. This is proved by induction on the derivation. We show the key case:
\[
\frac{\text{(cofix } x^{n+1} A^{n+1} b ; i A \Gamma \vdash x^{n+1} A^{n+1} A^{n+1} ; \text{type } i \geq n) } }{ \Gamma \vdash x^{n+1} [A]; i [A] } \quad (1)
\]

By Induction Hypothesis,
\[
\begin{align*}
[\Gamma], x^{n+1} [A] & \vdash b ; i [A] \quad (\text{1}) \\
[\Gamma] & \vdash [A]; i [A] \quad (2)
\end{align*}
\]
From the above rule, we know that \( i \geq n \). However, we cannot apply \( \text{cofix}^{n+1} \) unless \( i \geq n + 1 \). The trick is to apply Time Adjustment (Theorem 4.3) to (1) and (2).

\[
\begin{align*}
\Gamma, x_{i+2} : [A] &\triangleright b^{i+1} [A] \\
\Gamma &\triangleright b^{i+1} [A] : i+1
\end{align*}
\]

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

\[
\Gamma \triangleright b^{i+1} (\text{cofix} x : [A], [b]; i+1) [A]
\]

(3)

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

\[
\Gamma \triangleright b^{i+1} [b] [x := (\text{cofix} x : [A], [b]); i] [A]
\]

Since \( (\text{cofix} x : A).b \) is \( \text{cofix} \) and all the systems of the \( \lambda \)-cube extended with corecursion are \( \text{infinitary weak} \) \( \beta\sigma\gamma\)-normalizing.

**Theorem 6.7** (Infinitary Weak \( \beta\sigma\gamma \)-Normalization). If \( \lambda^\omega(S) \models \beta\sigma\gamma \)-WN then \( \lambda^\omega(S) \models \beta\sigma\gamma \)-WN\(^\infty\). Moreover, the infinitary weak \( \beta\sigma\gamma \)-normal forms are obtained in \( \omega \)-steps.

**Proof.** Suppose \( \Gamma \vdash a ; A \). Hence, \( \Gamma \vdash b ; a \). We show that there exists a normalizing strategy starting from \( a \). We construct a reduction sequence of following form:

\[
a = a_0 \rightarrow b \rightarrow a_1 \rightarrow b \rightarrow a_2 \ldots
\]

(4)

We define \( a_0 \) as \( a_0 = a \). By Theorem 6.6, we have that \( \Gamma \vdash b^{i+1} a_0 ; [A] \). Since \( \lambda^\omega(S) \models \beta\sigma\gamma \)-weakly normalizing, so is \( \lambda^\omega(S) \) for all \( n \). We can, then, define \( a_1 \) as the \( \beta\sigma\gamma \)-normal form of \( a_0 \). By Theorem 4.5, \( \Gamma \vdash b^{i+1} a_1 ; [A] \). We repeat this process for each \( n \). The reduction sequence (4) has the following form:

\[
a = a \rightarrow b \rightarrow a_1 \rightarrow b \rightarrow a_2 \rightarrow b \ldots
\]

(5)

where for all \( n \) there exist \( \Gamma_n \) and \( A_n \) such that \( \Gamma_n \vdash b^{i+1} a_n ; A_n \).

By Theorem 6.4, we have that \( (a_n) \) is in \( \beta\sigma\gamma \)-normal form for all \( n \geq i \). From \( i \) onwards, the sequence of truncations \( a_0, a_{i+1}, a_{i+2} \ldots \) is increasing (with respect to the subterm relation). It is clear that the reduction sequence (5) is strongly converging. Its limit \( a_n \) exists and it is in infinite \( \beta\sigma\gamma \)-normal form.

**Corollary 6.8.** \( \lambda^\omega(C) \) and all the systems of the \( \lambda \)-cube extended with corecursion are infinitary weakly \( \beta\sigma\gamma \)-normalizing.

**Proof.** It follows from Theorems 4.17 and 6.7, that \( \lambda^\omega(C) \) is infinitary weakly \( \beta\sigma\gamma \)-normalizing. Since all the systems of the \( \lambda \)-cube extended with corecursion are included in \( \lambda^\omega(C) \), we can conclude infinitary weakly \( \beta\sigma\gamma \)-normalizing 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 \( \gamma \)-normalizing.

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

For example, the term \( \lambda x : A.\text{zero} \vdash \Omega \) is infinitarily weakly \( \beta\sigma\gamma \)-normalizing but it is not infinitarily strongly \( \beta\sigma\gamma \)-normalizing.

**Definition 7.2** (Infinitary Strongly Normalizing CoPTS). Let \( \rho \) be a notion of reduction. We say that \( \lambda^\omega(S) \) is infinitarily strongly \( \rho \)-normalizing if for all \( a \in C \) such that \( \Gamma \vdash a ; A \) we have that \( a \) is strongly \( \rho \)-normalizing.

**Notation 7.3.** \( \lambda^\omega(S) \models \rho -\text{SN}^\infty \) if \( \lambda^\omega(S) \) is infinitarily strongly \( \rho \)-normalizing.

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

**Theorem 7.4** (Depth of Variables). Let \( \Gamma, x_1 : A, \Gamma' \vdash b ; B. \) Then the depth of all occurrences of \( x \) in \( b \) is greater than \( i - j \) if \( i > j \).

**Proof.** We have to prove it simultaneously with the statement: if \( \Gamma, x_1 : A, \Gamma' \vdash b ; B \) and \( y_1 : C \in \Gamma \) 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 \( \Gamma, x_1 : (\bullet A) \vdash b ; B \) then the depth of all occurrences of \( x \) in \( b \) is greater than 0.

**Corollary 7.6** (Depth of \( x \) in \( \text{cofix} \)). If \( \Gamma \vdash (\text{cofix} x : A).b ; A \) 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 \( \text{cardfix}(a) \) be the number of fixed points of \( a \) at depth \( n \).

**Theorem 7.7** (Strong Normalization of \( \gamma \)-reduction at depth \( n \)).\( \Gamma \vdash a; A \).\n
1. If \( a \rightarrow_{\gamma} b \) then \( \text{cardfix}(a) > \text{cardfix}(b) \).
2. Any reduction sequence of \( \gamma \)-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 \( a = a_0 \rightarrow_{\gamma} a_1 \rightarrow_{\gamma} a_2 \ldots \). From the first part, we would have an infinite decreasing sequence of natural numbers \( \text{cardfix}(a_0) > \text{cardfix}(a_1) > \ldots \) This is a contradiction. This means that this reduction sequence has to be finite.

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

**Proof.** Suppose there is an infinite reduction sequence starting from \( a = a_0 \rightarrow_{\gamma} a_1 \rightarrow_{\gamma} a_2 \ldots \) 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 an \( n \) such that from \( n \) 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 \( C^\infty \). Restricted to lambda terms, the equality relation induced by \( \beta\sigma\gamma \)-normal form —two terms are equivalent if 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.

**Comparison 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 \( \Theta \) can be typed and both have type \((\bullet A \rightarrow A) \rightarrow A \). Nakano proves that all typable terms have a Böhm tree without \( \perp \) which amounts to saying that they have an infinite \( \beta \)-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 \( C^\infty \)). For example, it can type \( Yf \) whose infinite normal form is the following:

\[
f (f \ldots )
\]
and also \( Y(λxy.yz) \) is typable using the recursive type \( μX.(X → B) → B \) whose infinite normal form is the following:

\[
(λy_1.y_1 (λy_2.y_2(λy_3.y_3...)))
\]

Krishnaswami and Benton’s typed lambda calculus of reactive programs use an equational theory instead of reduction [33]. Corollary 6.8 generalizes and strengthens in several directions the result in [33] where only weak normalization is proved for the fragment of \( λ^ω \) without fixed points.

Krishnaswami, Benton and Hoffman show another variant of \( λ^ω \) with linearity in [35]. 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 \( Δ \) is introduced that resembles a moded operator. The meaning of this operator is not the same as \( ▪.A \). While \( ▪.A \) can be understood as the information displayed in the future, \( Δ \) 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 \( Γ_1, Γ_2, ..., Γ_0 \) where \( Γ_i = \{x_1; A_1, ..., x_n; A_n\} \). Judgements in a MPTS can only infer types at time 0.

**Productivity.** The notion of productivity given in [16, 48] is equivalent to our notion of weak normalization. The notion of productivity is defined as weak normalization but excluding terms that do not contain constructors such as \((tl (tl (tl ...) ))\). In our case, we exclude terms without constructors from the start by defining an appropriate metric on terms.

The guardedness condition is a criterion that ensures productivity [10]. We cannot say that the guardedness condition is more restrictive than CoPTS. On one hand, CoPTS’s can type some terms that do not satisfy the guardedness condition as shown in Example 3.8. On the other hand, CoPTS’s cannot type the following example which satisfies the guardedness condition:

\[
pairup(a_1:a_2:xs) = cons(a_1, a_2) (pairup xs)
\]

The papers [16, 48] define a decidable criterion on (first order) term rewriting systems to ensure that programs using corecursive equations are productive [16, 48]. Using this criteria, the terms given in Example 3.8 would satisfy their criterion as well as the example of pairup above.

As pointed out by Eduardo Giménez in [21], the guardedness condition (this applies to the criterion in [16, 48] as well) has the problem of being a syntactic condition that can be checked only when the proof has been completed. So, it would be desirable to have a typing mechanism that prevents the user from doing a bad recursive call while she is writing the proof and not at the end.

**Other approaches to corecursion.** Hutton and Jaskelioff propose 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.

\[
zeros'''' = (cofix x : Ω → Stream Nat) .
(cocons 0 (interleave xs (tl x)) )
\]

The infinite normal form of zeros’’’’ is

\[
(cocons 0 (cocons 0 (cocons 0 ...)))
\]

**Techniques to prove normalization.** In order to prove preservation of strong normalization without contracting fixed points, we use two translations: one from \( λ^ω \) into \( λ_ω \) (Theorem 4.12) and another one from \( λ^ω(C) \) to \( λ^ω \) (Theorem 4.17). The translation from \( λ^ω(C) \) to \( λ^ω \) is an adaptation of the one given by Geuvers and Nederhof in [18]. This translation preserves reduction in a way that one step is mapped into one or more steps. The translation from \( λ^ω(C) \) to \( λ^ω \) codes the streams making use of polymorphism and ‘ignores’ the modality. As a consequence of this, \( σ\)-steps that contract the modal operator can be cancelled. 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 \( λS \models β-SN \) implies \( λ^ω(S) \models β-SN \) where \( λ^ω(S) \) is the extension of \( λ(S) \) with definitions where the translation can cancel \( δ\)-steps [43].

In order to prove infinitary weak normalization we used an auxiliary system \( \langle \rangle \) and the unfolding \( [a] \). This technique is used in [34] to prove that all typable terms have an \( m\)-normal form for a calculus based on \( λ^ω \) with linearity. The notion of \( m\)-normal form is defined in [34] in terms of the auxiliary system \( \langle \rangle \). This does not ensure yet that typable terms are well-behaved.

**9. Future Work**

Our work is closely related to the metric model introduced by Birkedal et al in [7] (used later by Krishnaswami and Benton in [33]). It will be interesting to define a Böhm model for corecursion on streams by interpreting terms as infinite normal forms [6, 27]. 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, 33] and the topos of trees [8].

To ensure that the Böhm model is well defined, we need to prove infinitary confluence besides infinitary weak normalization. The problem is that \( \rightarrow_{βσγ} \) is not confluent on untypable terms. We construct a counter-example from the \( σ\)-rules which are hypercollapsing, i.e. they are of the form \( C[x] → x \) [28, 29].

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

\[
Ω' \\
Ω
\]

\[
(\text{tl (cons 0 (tl (cons 0 ... ))}))
\]

\[
γ \quad \text{cannot be joined. The terms } Ω \text{ and } (\text{tl (cons 0 (tl (cons 0 ... ))}))\text{ can only reduce to themselves.}
\]

Ketema and Simonsen prove confluence up to hypercollapsing terms for orthogonal infinitary combinatory reduction systems. However, we cannot apply their result. This is because \( C^∞ \) 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 \( λ^ω(S) \) is infinitary strongly \( γ\)-normalizing. However, it remains open to prove it for \( βσγ \).

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, 47]. This will allow us to capture other notions of infinite data apart from streams such as infinite trees or equality between infinite objects.

Consider a basic primitive recursive function such as \( + \) defined as follows:

\[
+ = (λx : \text{Nat}. \text{cofix } f : (\text{Nat} → \text{Nat}). \lambda y : \text{Nat}. \begin{cases}
\text{case } y \text{ is } 0 \quad \text{then } x \quad \text{is} \quad \text{succ } z \text{ then succ } (f z)
\end{cases}
\]

\[
\text{cofix } f : (\text{Nat} → \text{Nat}). \lambda y : \text{Nat}. \begin{cases}
\text{case } y \text{ is } 0 \quad \text{then } x \quad \text{is} \quad \text{succ } z \text{ then succ } (f z)
\end{cases}
\]

\[\begin{align*}
\text{cofix } f : (\text{Nat} → \text{Nat}). \lambda y : \text{Nat}. \begin{cases}
\text{case } y \text{ is } 0 \quad \text{then } x \quad \text{is} \quad \text{succ } z \text{ then succ } (f z)
\end{cases}
\end{align*}\]
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].

The metrics considered for infinitary rewriting are smaller than ours [28–30]. 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.

We have not added $\eta$-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 $\lambda x. \beta x. (\lambda y. B. y)x$ for $A \neq B$ [41]. 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 Petrisan for discussing the papers by Krishnaswami and Benton with us. We would also like to thank Neelakantan Krishnaswami for a helpful email exchange. Moreover, we would like to thank the reviewers for the detailed and helpful comments and suggestions that they provided.

References


