A Coinductive Representation of Computable Functions

Alvin Tang
The Australian National University, Canberra, Australia

Dirk Pattinson
The Australian National University, Canberra, Australia

Abstract

We investigate a representation of computable functions as total functions over 2^∞, the set of finite and infinite sequences over $\{0, 1\}$. In this model, infinite sequences are interpreted as non-terminating computations whilst finite sequences represent the sum of their digits. We introduce a new definition principle, *function space corecursion*, that simultaneously generalises minimisation and primitive recursion. This defines the class of *computable corecursive functions* that is closed under composition and function space corecursion. We prove computable corecursive functions represent all partial recursive functions, and show that all computable corecursive functions are indeed computable by translation into the untyped λ-calculus.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Streaming models; Theory of computation → Recursive functions

Keywords and phrases Computability, Coinduction

Digital Object Identifier 10.4230/LIPIcs.CALCO.2025.7

Acknowledgements The term "hiaton" to describe a token that symbolises a computational step without visible observations was suggested by Thomas Streicher.

1 Introduction

Partial recursive functions as a model of computation intuitively operate on natural numbers, and therefore have been used as a foundation of recursion theory. A notable drawback, however, is that they are partial – function values may be undefined. This complicates their use for formal reasoning, especially with the lack of support for partial functions in many interactive theorem provers [8], and the embedding of types into partial (recursive) functions require restrictions to assert well-definedness of values [15].

Coprogramming [2, 7] has been proposed as a way to deal with non-terminating computations. The main idea here is that functions progressively reveal an increasing part of their possibly infinite output. There are plenty of models tailored to specific programming paradigms and even type theories to ensure the coherence of corecursive function definitions [13]. Partial recursive functions, on the other hand, appear to be on the opposite side of the spectrum as they do not produce infinite function values – the result of an infinite computation is simply undefined.

Conceptually, the partiality in the theory of recursive functions arises from the minimisation operator. Given a function f, we compute μf by successively evaluating $f(0), f(1), \ldots$ and return n once we find that $f(n) = 0$. Nonetheless, once it is known that $f(0) > 0$, we can be sure μf must be no less than 1 if it is defined. Using a unary representation of natural numbers, it is already safe to emit the digit 1. This process can then be repeated for $n = 2, 3, \ldots$ and emit more 1's whenever we find that $f(n) > 0$. Once we find that $f(n) = 0$, the computation terminates.

This leads to the idea of regaining totality by representing partial recursive functions as functions that operate on the set of finite and infinite sequences of 1's, where $1^\omega = (1, 1, 1, \dots)$ represents the result of an infinite computation, such as the case for μf where $f(n) > 0$ for all $n \in \mathbb{N}$.

This unary representation of natural numbers, albeit an intuitive coinductive data structure, would not suffice for expressing all computations, at least if we aim for computable functions to be closed under composition. Furthermore, it is necessary to consider the implications when the infinite sequence 1^ω is taken as a function argument. As an example, consider the function even that determines the parity of its argument, where we represent false by the empty sequence and true by any non-empty sequence. In order to be total, the computation of $\mathsf{even}(1^\omega)$ must, after finitely many steps, either emit the empty sequence as a result and terminate, whence $\mathsf{even}(1^\omega) = \mathsf{false}$, or output an initial digit 1, which indicates $\mathsf{even}(1^\omega) = \mathsf{true}$. At this point, even will have only read a finite section of its input, say the first k digits. This entails $\mathsf{even}(1^\omega) = \mathsf{even}(1^k) = \mathsf{even}(1^{k+1})$, which is an obvious contradiction. This demonstrates even does not have a representation as a total computable function on the set of finite and infinite sequences of 1's. In fact, this exemplifies that "computable functions are continuous" [20, page 73].

We solve this problem by considering computations over 2^∞, the set of finite and infinite sequences over $\{0, 1\}$. Here, the digit 0 represents a *hiaton* that can be read as "we do not know yet". The computation of $\mathsf{even}(1^\omega)$ would then be expected to yield the value 0^ω, consistent with the intuition that at any finite stage of computing $\mathsf{even}(1^\omega)$, the result is yet unknown.

This provides the starting point of this computation model – considering functions as programs operating on the coinductive data structure of binary sequences. Our main contribution is a new definition principle that we call *function space corecursion*, allowing us to represent precisely all partial recursive functions. We show that function space corecursion can be used to express both primitive recursion and minimisation. The interpretation of binary sequences is formally defined using mixed induction-coinduction, which is essential for correlating the behaviour of these corecursive functions with that of partial recursive functions.

Analogous to partial recursive functions, the class of *computable corecursive functions* contains the zero, successor and projection functions, and is closed under composition and function space corecursion. A principal result of this work is the representation theorem proving the Turing completeness of this model. This involves the completeness theorem which proves that any computable program can be represented in this functional model, and the soundness theorem asserting the fact that every computable corecursive function is indeed computable by encoding them in the untyped λ-calculus.

Whilst this paper focuses on the (co)algebraic foundations of the corecursive model and its expressive power, it opens up a large number of new lines of investigation, some of which are discussed in the conclusion.

Related Work

The interpretation of computable functions over the set 2^∞ is closely related to *lazy* computation, which intuitively allows partial outputs to be emitted before the end result of the computation is known. More formally, identifying whether two sequences have the same number of 1's leads to the lazy natural numbers, and the aspects of computability have been studied in [1, 9]. Function space corecursion, the key principle this paper contributes, is nevertheless novel. Computation on possibly infinite sequences is the domain of type-2 Turing machines [19] but the interpretation is different – each sequence is a *representation* of a point in a topological space, rather than a trace of an intensional computation process. Infinite-time Turing machines [10] are also different as they are machine models of computation where ordinal-many steps can be taken.

2 Preliminaries and Notation

We use the standard set-theoretic notation and basic facts about algebras and coalgebras of set-endofunctors that do not go beyond the tutorial on (co)algebras and (co)induction [11]. We write $\times$ for the Cartesian product, and $+$ for the coproduct in the category of sets and functions. Our universal algebra is similarly basic and is covered by [4, 6, 18]. In particular, we use algebraic signatures (i.e., sets of function symbols with arities) and write $T_\Sigma(X)$ for the set of terms with variables in X that can be constructed from a signature Σ.

We presuppose familiarity with partial recursive functions (see [16]) that we conceptualise as the least class that contains the constant zero, the unary successor function (suc), all k-ary projections to the i^{th} element (π_i^k) and is closed under primitive recursion as well as minimisation. We write $f(x_1, \dots, x_k) \downarrow$ if $f(x_1, \dots, x_k)$ is defined, and $f(x_1, \dots, x_k) \uparrow$ otherwise. To be precise, we take the composition of a partial function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ with a tuple $\langle g_1, \dots, g_n \rangle$ of k-ary partial recursive functions to be the function $f \circ \langle g_1, \dots, g_n \rangle : \mathbb{N}^k \rightharpoonup \mathbb{N}$ where $f \circ \langle g_1, \dots, g_n \rangle(x_1, \dots, x_k) = f(g_1(x_1, \dots, x_k), \dots, g_n(x_1, \dots, x_k))$ if all of $g_1(x_1, \dots, x_k), \dots, g_n(x_1, \dots, x_k)$ are defined, and otherwise undefined. Note that even if all $g_i(x_1, \dots, x_k)$ are defined, f might be undefined at $(g_1(x_1, \dots, x_k), \dots, g_n(x_1, \dots, x_k))$. In this case, $f \circ \langle g_1, \dots, g_n \rangle(x_1, \dots, x_k) \uparrow$ holds. Similarly, if $g : \mathbb{N}^k \rightharpoonup \mathbb{N}$ and $h : \mathbb{N}^{k+2} \rightharpoonup \mathbb{N}$ are partial recursive functions, $\mathsf{PR}(g, h) : \mathbb{N}^{k+1} \rightharpoonup \mathbb{N}$ is given by:

$$\begin{aligned} \mathsf{PR}(g,h)(0, x_1, \dots, x_k) &= g(x_1, \dots, x_k) \\ \mathsf{PR}(g,h)(n+1, x_1, \dots, x_k) &= h(n, \mathsf{PR}(g,h)(n, x_1, \dots, x_k), x_1, \dots, x_k) \end{aligned}$$

in case all arguments of h above are defined.

A similar caveat applies to minimisation, where $\mu(f)(x_1, \dots, x_k) = n$ if:

- $f(m, x_1, \dots, x_k) \downarrow$ and $f(m, x_1, \dots, x_k) > 0$ for all $m = 0, \dots, n-1$, and
- $f(n, x_1, \dots, x_k) = 0$.

Our conventions for the untyped λ-calculus, including natural numbers and if-then-else conditionals, follow Barendregt's encoding [3]. We write Λ for the set of all untyped λ-terms and denote β-reduction by $\longrightarrow_\beta$. Two λ-terms are considered equal by β-equality.

The empty sequence is denoted by $[\,]$. If A is a set, we write A^n for the n-fold Cartesian product of A, A^* for the set of finite sequences with elements in A, and A^ω for the set of infinite sequences over A. The set of finite and infinite sequences with elements in A is denoted by $A^\infty = A^* \cup A^\omega$. For $\alpha \in A^*$, we write α^n for the n-fold concatenation of α with itself, and α^ω for the infinite sequence $\alpha \cdot \alpha \cdot \alpha \dots$ that arises by concatenating α with itself countably many times.

3 Natural Number and Function Representations

To establish the relationship between partial recursive functions and the new corecursive model, the representation of natural numbers and functions operating on them is formally defined with respect to binary sequences.

▶ **Definition 1** (Coinductive Representation of Numbers). *We write $2 = \{0, 1\}$ for the set of binary digits, and $\overline{\mathbb{N}} = 2^\infty$ for the set of finite and infinite sequences over 2 that we call the* intensional natural numbers*. For $\alpha \in \overline{\mathbb{N}}$, we write $\alpha \uparrow$ if α is infinite, and $\alpha \downarrow$ if $\alpha \in \{0,1\}^*$ is finite. A finite intensional natural number $\alpha = (a_0, \dots a_k) \in 2^*$* represents *the number $n \in \mathbb{N}$ if $\sum_{i=0}^{k} a_i = n$, and we write $\alpha \ \underline{\mathsf{r}}\ n$ in this case. A function $f : \overline{\mathbb{N}}^k \to \overline{\mathbb{N}}$* represents *a partial numeric function $\hat{f} : \mathbb{N}^k \rightharpoonup \mathbb{N}$ if*

- *$f(\alpha_1, \dots, \alpha_k) \downarrow$ iff $\hat{f}(n_1, \dots, n_k) \downarrow$ whenever each $\alpha_i \ \underline{\mathsf{r}}\ n_i$ for $i = 1, \dots, k$, and*
- *$f(\alpha_1, \dots, \alpha_k) \ \underline{\mathsf{r}}\ \hat{f}(n_1, \dots, n_k)$ in this case.*

Clearly, every natural number has countably many representations, and a finite sequence represents a terminating computation. We think of infinite sequences as non-terminating computations, but do not identify them with a single symbol or value, as they can be non-terminating in many different ways. The sequence $(1, 1, 1, \dots)$ intuitively represents a point at infinity, whereas $(0, 0, 0, \dots)$ is a non-terminating computation that never reveals any information.

It is well known that $\overline{\mathbb{N}}$ arises as the final coalgebra for the set-endofunctor $FX = 1 + 2 \times X$ (see e.g., [11]). As such, it supports coinductive definitions and proof principles. In particular, we are going to use *primitive corecursion* in the sequel.

▶ **Proposition 2.** *Let $\sigma : \overline{\mathbb{N}} \to 1 + 2 \times \overline{\mathbb{N}}$ be the structure map of the final coalgebra with $FX = 1 + 2 \times X$ and let C be a set. Then, for all $\phi : C \to F(C + \overline{\mathbb{N}})$, there exists a unique $f : C \to \overline{\mathbb{N}}$ such that $\sigma \circ f = F[f, \mathsf{id}] \circ \phi$.*

For the proof, see [17]. The unique function u in the proposition above is known as an *apomorphism* in the literature. Spelling this out, this means that for every pairwise disjoint partition $C = C_0 \cup C_1 \cup C_2$ and all functions $f_0 : C_0 \to C$, $f_1 : C_1 \to C$ and $f_2 : C_2 \to \overline{\mathbb{N}}$, there is a uniquely defined function $u : C \to \overline{\mathbb{N}}$ that satisfies the equations on the left below.

$$
\begin{aligned}
u(c) &= 0 : u(f_0(c)) \text{ if } c \in C_0 \qquad & \mathsf{add}(0 : \alpha, \beta) &= 0 : \mathsf{add}(\alpha, \beta) \\
u(c) &= 1 : u(f_1(c)) \text{ if } c \in C_1 & \mathsf{add}(1 : \alpha, \beta) &= 1 : \mathsf{add}(\alpha, \beta) \\
u(c) &= f_2(c) \text{ if } c \in C_2 & \mathsf{add}([\,], \beta) &= \beta
\end{aligned}
$$

This allows us to define a version of addition for intensional natural numbers by concatenating both sequences by the equations on the right. We note that the addition defined above is extensional in the sense that if arguments just differ in zeros, the same is true for the result.

$$\begin{tikzcd}
C \arrow[r, "f"] \arrow[d, "\phi"'] & \overline{\mathbb{N}} \arrow[d, "\sigma"] \\
1 + 2 \times (C + \overline{\mathbb{N}}) \arrow[r, "{F[f,\mathsf{id}]}"'] & 1 + 2 \times \overline{\mathbb{N}}
\end{tikzcd}$$

Figure 1 Commutative diagram for the apomorphism with $F\overline{\mathbb{N}} = 1 + 2 \times \overline{\mathbb{N}}$.

More formally, we have the following notion of extensional equivalence.

▶ **Definition 3** (Extensional equivalence). *The relation $\simeq$ of* extensional equivalence *is given as the mixed fixpoint defined by the inductive rules (indicated with single-line fractions) and the co-inductive rules (written with double-line fractions) below where $\alpha, \beta \in \overline{\mathbb{N}}$.*

$$\frac{}{[\,] \simeq [\,]} \qquad \frac{\alpha \simeq \beta}{0 : \alpha \simeq \beta} \qquad \frac{\alpha \simeq \beta}{\alpha \simeq 0 : \beta} \qquad \frac{\alpha \simeq \beta}{0 : \alpha \simeq 0 : \beta} \qquad \frac{\alpha \simeq \beta}{1 : \alpha \simeq 1 : \beta}$$

More precisely, extensional equivalence is the mixed fixpoint $\nu R.\mu S.I(S) \cup C(R)$ such that

$$I(S) = \{(\alpha, \beta) \mid (\alpha, \beta) \text{ is a conclusion of an inductive rule with premisses in } S\}$$

and $C(R)$ as the analogously defined operator using the coinductive rules, where $\nu X.O(X)$ and $\mu X.O(X)$ are the greatest and least fixpoints of a monotone operator O.

We say that a function $f : \overline{\mathbb{N}}^k \to \overline{\mathbb{N}}$ is extensional if $f(\alpha_1, \dots, \alpha_k) \simeq f(\beta_1, \dots, \beta_k)$ whenever $\alpha_i \simeq \beta_i$ for $i = 1, \dots, k$, and a functional $\Phi : (\overline{\mathbb{N}}^k \to \overline{\mathbb{N}}) \to (\overline{\mathbb{N}}^k \to \overline{\mathbb{N}})$ is extensional if $\Phi(f)$ is extensional whenever f is extensional.

One way to read this definition is that two intensional natural numbers are extensionally equal if there is a possibly infinite derivation, using both inductive and coinductive rules, where there are only finitely many applications of inductive rules between any two coinductive rules. For finite sequences, extensionally equal numbers represent the same natural numbers.

▶ **Proposition 4.** *Let $\alpha, \beta \in \overline{\mathbb{N}}$ with $\alpha \simeq \beta$. Then, α is finite if and only if β is finite. Moreover, if $\alpha \ \underline{\mathsf{r}}\ n$ and $\beta \ \underline{\mathsf{r}}\ k$ for finite α and $\beta \in \overline{\mathbb{N}}$ and $n, k \in \mathbb{N}$, then $n = k$.*

Proof. The sequence α is finite only if the inductive and coinductive rules are applied finitely many times and ultimately the axiom $\alpha = [\,] \simeq [\,] = \beta$ is applicable. This is analogous when β is finite.

Assuming $\alpha \simeq \beta$ for finite $\alpha, \beta \in \overline{\mathbb{N}}$, the coinductive rule stipulating $\alpha \simeq \beta$ implies $1 : \alpha \simeq 1 : \beta$ is applied finitely many times. The same number of 1's must be observed in α and β. Hence, α and β must represent the same natural number. ◀

For extensional functions, we can safely use the intensional natural numbers interchangeably with any of the corresponding binary sequences.

4 Function Space Corecursion

Function space corecursion is a definition principle similar to primitive recursion which allows us to generalise both primitive recursion and minimisation operating on intensional natural numbers. The intuition is best explained in comparison to primitive recursion. A definition of $f = \mathsf{PR}(g, h) : \mathbb{N}^{k+1} \rightharpoonup \mathbb{N}$ by primitive recursion relies on two auxiliary (or already defined) functions $g : \mathbb{N}^k \rightharpoonup \mathbb{N}$ (that defines f when the first argument is zero), and $h : \mathbb{N}^{k+2} \rightharpoonup \mathbb{N}$ (that is used in the successor case).

Function space corecursion defines a function f of type $\overline{\mathbb{N}}^k \to \overline{\mathbb{N}}$ in terms of $k+1$ functions $(f_0, \dots, f_k)$ that are all of the arity k. The first function f_0 yields the result if the first argument is the empty sequence $[\,] \in \overline{\mathbb{N}}$:

$$f([\,], \alpha_2, \dots, \alpha_k) = f_0([\,], \alpha_2, \dots, \alpha_k).$$

The remaining functions simultaneously transform all arguments in a recursive call, where $\mathsf{suc}\ \alpha = 1 : \alpha$:

$$f(\mathsf{suc}\ \alpha_1, \alpha_2, \dots, \alpha_k) = 0 : f(f_1(\alpha_1, \dots, \alpha_k), \dots, f_k(\alpha_1, \dots, \alpha_k))$$

The crucial point of function space corecursion, and its main difference to primitive recursion, is that all functions $f_0, \dots, f_k$ *can change in every recursive call.* That means, we stipulate the equations

$$\begin{aligned}
\mathsf{FCR}(f_0, \dots, f_k)([\,], \alpha_2, \dots, \alpha_k) &= f_0([\,], \alpha_2, \dots, \alpha_k)\\
\mathsf{FCR}(f_0, \dots, f_k)(0 : \alpha_1, \alpha_2, \dots, \alpha_k) &= 0 : \mathsf{FCR}(f_0, \dots, f_k)(\alpha_1, \dots \alpha_k)\\
\mathsf{FCR}(f_0, \dots, f_k)(1 : \alpha_1, \alpha_2, \dots, \alpha_k) &= 0 : \mathsf{FCR}(f'_0, \dots, f'_k)(\alpha'_1, \dots \alpha'_k)
\end{aligned}$$

where $\alpha'_i = f_i(\alpha_1, \dots, \alpha_k)$ and $f'_0, \dots, f'_k$ are the changed versions of $f_0, \dots, f_k$. We express this change by postulating that $f'_i = \Phi_i(f_i)$ for an operator $\Phi_i : (\bar{\mathbb{N}}^k \to \bar{\mathbb{N}}) \to (\bar{\mathbb{N}}^k \to \bar{\mathbb{N}})$. The functionals $\Phi_0, \dots, \Phi_k$ then become parameters of a definition by function space corecursion. The following definition makes this precise in terms of intensional natural numbers.

▶ **Definition 5** (Function space corecursion). *Let $k > 0$ and fix functionals $\Phi_0, \dots, \Phi_k : (\bar{\mathbb{N}}^k \to \bar{\mathbb{N}}) \to (\bar{\mathbb{N}}^k \to \bar{\mathbb{N}})$. We say that the function $\mathsf{FCR}(\Phi_0, \dots, \Phi_k) : (\bar{\mathbb{N}}^k \to \bar{\mathbb{N}})^{k+1} \to \bar{\mathbb{N}}^k \to \bar{\mathbb{N}}$ is defined by*

$$\begin{aligned}
\mathsf{FCR}\,\vec{\Phi}\,\vec{f}(\quad [\,], \vec{\alpha}) &= f_0([\,], \vec{\alpha})\\
\mathsf{FCR}\,\vec{\Phi}\,\vec{f}(0 : \alpha_1, \vec{\alpha}) &= 0 : \mathsf{FCR}\,\vec{\Phi}\,\vec{f}(\alpha_1, \vec{\alpha})\\
\mathsf{FCR}\,\vec{\Phi}\,\vec{f}(1 : \alpha_1, \vec{\alpha}) &= 0 : \mathsf{FCR}\,\vec{\Phi}\,\vec{f}'\,(f_1(\alpha_1, \vec{\alpha}), \dots, f_k(\alpha_1, \vec{\alpha}))
\end{aligned}$$

where $\vec{\Phi} = (\Phi_0, \dots, \Phi_k)$, $\vec{f} = (f_0, \dots, f_k)$, $\vec{\alpha} = (\alpha_2, \dots, \alpha_k)$ and $\vec{f}' = (f'_0, \dots, f'_k)$ such that $f'_j = \Phi_j(f_j)$ is defined by function space corecursion *using the functionals $\Phi_0, \dots, \Phi_k$ and initial functions $f_0, \dots, f_k$.*

▶ Remark 6 (Variations). There are many different possible ways to define function space corecursion. Some variations are to limit the number of functionals, possibly hard-coding the transformation of some of the arguments or being more liberal about the arities of the f_i. It may also allow Φ_i to depend on all of $f_0, \dots, f_k$, i.e., to change the type of Φ_i to $(\bar{\mathbb{N}}^k \to \bar{\mathbb{N}})^{k+1} \to (\bar{\mathbb{N}}^k \to \bar{\mathbb{N}})$. Another variation is *isotone function space corecursion* defined like function space corecursion, but with the last line (and the name) changed to

$$\mathsf{ICR}\,\vec{\Phi}\,\vec{f}(1 : \alpha_1, \vec{\alpha}) = 1 : \mathsf{ICR}\,\vec{\Phi}\,\vec{f}'\,(f_1(\alpha_1, \vec{\alpha}), \dots, f_k(\alpha_1, \vec{\alpha}))$$

whilst the remaining clauses remain the same as in Definition 5. This operator is isotone, as it evidently increases the number of 1's in the function value, but clearly not complete in the sense of being able to define all partial recursive functions. The constant zero function, for instance, is not definable. Other variations include mixed function space corecursion that combines both plain and isotone function space corecursion. Here we stipulate that:

$$\begin{aligned}
\mathsf{MCR}\,\vec{\Phi}\,\vec{f}(\quad [\,], \quad \beta, \vec{\gamma}) &= \quad f_0([\,], \beta, \vec{\gamma})\\
\mathsf{MCR}\,\vec{\Phi}\,\vec{f}(0 : \alpha, \quad \beta, \vec{\gamma}) &= 0 : \mathsf{MCR}\,\vec{\Phi}\,\vec{f}(\alpha, \beta, \vec{\gamma})\\
\mathsf{MCR}\,\vec{\Phi}\,\vec{f}(\quad \alpha, 0 : \beta, \vec{\gamma}) &= 0 : \mathsf{MCR}\,\vec{\Phi}\,\vec{f}(\alpha, \beta, \vec{\gamma})\\
\mathsf{MCR}\,\vec{\Phi}\,\vec{f}(1 : \alpha, \quad [\,], \vec{\gamma}) &= 1 : \mathsf{MCR}\,\vec{\Phi}\,\vec{f}'\,(f_1(\alpha, [\,], \vec{\gamma}), \dots, f_{k+2}(\alpha, [\,], \vec{\gamma}))\\
\mathsf{MCR}\,\vec{\Phi}\,\vec{f}(1 : \alpha, 1 : \beta, \vec{\gamma}) &= 0 : \mathsf{MCR}\,\vec{\Phi}\,\vec{f}'\,(f_1(\alpha, \beta, \vec{\gamma}), \dots, f_{k+2}(\alpha, \beta, \vec{\gamma}))
\end{aligned}$$

The second argument controls whether a non-zero digit will be emitted, and MCR generalises both FCR and ICR by fixing the second argument. We mainly focus on FCR as this suffices to represent all partial recursive functions. Nevertheless, all our arguments are easily adapted to MCR.

We make two simple observations before we turn to (lots of) examples. First, function space corecursion is indeed uniquely defined.

▶ **Proposition 7.** *The function* $\mathsf{FCR}\,\vec{\Phi}$ *is total and uniquely defined by the equations in Definition 5.*

Proof. We use Proposition 2. Let $C = ((\bar{\mathbb{N}}^k \to \bar{\mathbb{N}}) \to (\bar{\mathbb{N}}^k \to \bar{\mathbb{N}}))^{k+1} \times (\bar{\mathbb{N}}^k \to \bar{\mathbb{N}})^{k+1} \times \bar{\mathbb{N}}^k$, which is the uncurried domain of FCR. Suppose $\vec{\Phi} = (\Phi_0, \dots, \Phi_k)$, $\vec{f} = (f_0, \dots, f_k)$ and $\vec{f'} = (f'_0, \dots, f'_k)$ where $f'_i = \Phi_i(f_i)$. Let $\phi : C \to 1 + 2 \times (C + \bar{\mathbb{N}})$ be defined as follows:

$$\phi(\vec{\Phi}, \vec{f}, [\,], \vec{\alpha}) = \begin{cases} \mathrm{in}_L(*) & \text{if } f_0(\vec{\alpha}) = [\,] \\ \mathrm{in}_R(0, \mathrm{in}_R(\alpha')) & \text{if } f_0(\vec{\alpha}) = 0 : \alpha' \\ \mathrm{in}_R(1, \mathrm{in}_R(\alpha')) & \text{if } f_0(\vec{\alpha}) = 1 : \alpha' \end{cases}$$

$$\phi(\vec{\Phi}, \vec{f}, 0 : \alpha_1, \vec{\alpha}) = \mathrm{in}_R(0, \mathrm{in}_L(\vec{\Phi}, \vec{f}, \alpha_1, \vec{\alpha}))$$
$$\phi(\vec{\Phi}, \vec{f}, 1 : \alpha_1, \vec{\alpha}) = \mathrm{in}_R(0, \mathrm{in}_L(\vec{\Phi}, \vec{f'}, \alpha'_1, \vec{\alpha}'))$$

where $\vec{\alpha}' = (\alpha'_2, \dots, \alpha'_k)$ and $\alpha'_i = f'_i(\alpha_1, \vec{\alpha})$ for any $i = 1, \dots, k$. Proposition 2 now guarantees the unique existence of a function that satisfies the equations stipulated in Definition 5. ◀

The proof readily adapts to show the uniqueness of ICR and MCR where $C_1 \neq \emptyset$. The fact that 0's in the definition of FCR are just passed through and do not affect the computation implies that functions defined using function space corecursion are always extensional (Definition 3).

▶ **Proposition 8.** *If* $k > 0$ *and all of* $\Phi_0, \dots, \Phi_k$ *and* $f_0, \dots, f_k$ *are extensional, then so is* $\mathsf{FCR}\,\vec{\Phi}\,\vec{f}$ *where* $\vec{\Phi} = (\Phi_0, \dots, \Phi_k)$ *and* $\vec{f} = (f_0, \dots, f_k)$*.*

Proof. Let $B = \{(\alpha, \beta) \mid \alpha, \beta \in \bar{\mathbb{N}} \text{ and } \alpha \simeq \beta\}$ be the set of all extensional equivalence relations. Consider the set

$$E = B \cup \{(\mathsf{FCR}\,\vec{\Phi}\,\vec{f}\,\vec{\alpha}, \mathsf{FCR}\,\vec{\Phi}\,\vec{f}\,\vec{\beta}) \mid \vec{\alpha} \simeq \vec{\beta}\}$$

where $\vec{\Phi} = (\Phi_0, \dots, \Phi_k)$, $\vec{f} = (f_0, \dots, f_k)$, $\vec{\alpha} = (\alpha_1, \dots, \alpha_k)$ and $\vec{\beta} = (\beta_1, \dots \beta_k)$. The goal is to prove $E \subseteq B$. By the Knaster-Tarski theorem [12], it suffices to show $E \subseteq \mu S.I(S) \cup C(E)$ with I and C as in Definition 3. Let $(\gamma, \delta) \in E$. This proof goal clearly holds if $(\gamma, \delta) \in B$.

Now consider $\gamma = \mathsf{FCR}\,\vec{\Phi}\,\vec{f}\,\vec{\alpha}$ and $\delta = \mathsf{FCR}\,\vec{\Phi}\,\vec{f}\,\vec{\beta}$ with $\vec{\alpha} \simeq \vec{\beta}$, that is, $\alpha_i \simeq \beta_i$ for all $i = 1, \dots, k$. We have $B = \mu S.I(S) \cup C(B)$, that is, we can perform induction on the inductive part of extensional equivalence $\alpha_1 \simeq \beta_1$ to show that $(\gamma, \delta) \in \mu S.I(S) \cup C(E)$, with $b : \alpha_1 \simeq 1 : \beta$ for $b \in \{0, 1\}$ and $\alpha_1 \simeq \beta_1$ as base cases in addition to $\alpha_1 = \beta_1 = [\,]$.

If $\alpha_1 = \beta_1 = [\,]$, we have $\mathsf{FCR}\,\vec{\Phi}\,\vec{f}\,\vec{\alpha} = f_0(\vec{\alpha})$ and $\mathsf{FCR}\,\vec{\Phi}\,\vec{f}\,\vec{\beta} = f_0(\vec{\beta})$. By assumption, f_0 is extensional so $(f_0(\vec{\alpha}), f_0(\vec{\beta})) \in E$ as required.

If $\alpha_1 = 0 : \alpha'_1$ and $\beta_1 = 0 : \beta'_1$ with $\alpha'_1 \simeq \beta'_1$, we have

$$(\gamma, \delta) = (0 : \mathsf{FCR}\,\vec{\Phi}\,\vec{f}\,(\alpha'_1, \alpha_2, \dots, \alpha_k), 0 : \mathsf{FCR}\,\vec{\Phi}\,\vec{f}\,(\beta'_1, \beta_2, \dots, \beta_k)) \in C(E)$$

by definition of FCR and E. Similarly, if $\alpha_1 = 1 : \alpha'_1$ and $\beta_1 = 1 : \beta'_1$ with $\alpha'_1 \simeq \beta'_1$, we have

$$(\gamma, \delta) = (0 : \mathsf{FCR}\,\vec{\Phi}\,\vec{f'}\,(\tilde{\alpha}'_1, \tilde{\alpha}_2, \dots, \tilde{\alpha}_k), 0 : \mathsf{FCR}\,\vec{\Phi}\,\vec{f'}\,(\tilde{\beta}'_1, \tilde{\beta}_2, \dots, \tilde{\beta}_k)) \in C(E)$$

with $f'_i = \Phi_i(f_i)$ for $i = 0, \dots, k$, $\tilde{\alpha}'_i = f_i(\alpha'_1, \alpha_2, \dots, \alpha_k)$, and $\tilde{\beta}'_i = f_i(\beta'_1, \beta_2, \dots, \beta_k)$. By definition of FCR as well as the extensionality of Φ_i and f_i, this implies $(\gamma, \delta) \in C(E)$.

If $\alpha_1 = 0 : \alpha'_1$, we have by induction hypothesis that

$$(\mathsf{FCR}\,\vec{\Phi}\,\vec{f}\,(\alpha'_1, \alpha_2, \dots, \alpha_k), \mathsf{FCR}\,\vec{\Phi}\,\vec{f}\,(\beta_1, \beta_2, \dots, \beta_k)) \in \mu S.I(S) \cup C(E).$$

Unfolding the definition of FCR, and applying an inductive rule, this also implies

$$(0 : \mathsf{FCR}\,\vec{\Phi}\,\vec{f}\,(\alpha_0', \alpha_1, \ldots, \alpha_k), \mathsf{FCR}\,\vec{\Phi}\,\vec{f}\,(\beta_0, \beta_1, \ldots, \beta_k)) \in \mu S.I(S) \cup C(E).$$

The case where $\beta_0 = 0 : \beta_0'$ is symmetric. ◀

▶ **Example 9.** The addition of intensional natural numbers can be defined in two different ways. Using isotone function space corecursion, we stipulate $f(\alpha, \beta) = \beta$, $g(\alpha, \beta) = \alpha$ and $h(\alpha, \beta) = \beta$. With the identity functionals $\Phi = \Psi = \Gamma = \mathsf{Id} : (\overline{\mathbb{N}}^2 \to \overline{\mathbb{N}}) \to (\overline{\mathbb{N}}^2 \to \overline{\mathbb{N}})$ and $\mathsf{add} = \mathsf{ICR}(\Phi, \Psi, \Gamma)(f, g, h)$, we have:

$$\begin{aligned}
\mathsf{add}([\,], \beta) &= \mathsf{ICR}(\Phi, \Psi, \Gamma)(f, g, h)([\,], \beta) = f([\,], \beta) = \beta \\
\mathsf{add}(0 : \alpha, \beta) &= \mathsf{ICR}(\Phi, \Psi, \Gamma)(f, g, h)(0 : \alpha, \beta) \\
&\simeq \mathsf{ICR}(\Phi, \Psi, \Gamma)(f, g, h)(\alpha, \beta) = \mathsf{add}(\alpha, \beta) \\
\mathsf{add}(1 : \alpha, \beta) &= \mathsf{ICR}(\Phi, \Psi, \Gamma)(f, g, h)(1 : \alpha, \beta) \\
&= 1 : \mathsf{ICR}(\Phi, \Psi, \Gamma)(f, g, h)(g(\alpha, \beta), h(\alpha, \beta)) = 1 : \mathsf{add}(\alpha, \beta).
\end{aligned}$$

In other words, add is the coinductive version of the usual definition of natural numbers addition. It resembles the primitive recursive addition function $Add : \mathbb{N}^2 \rightharpoonup \mathbb{N}$ where $Add(0, m) = m$ and $Add(n + 1, m) = 1 + Add(n, m)$ by recursion on the first argument.

Alternatively, we can use the plain function space corecursion where occurrences of 1's in the first argument are accumulated in the second argument in every recursive call. That is, we consider $\Phi = \Psi = \Gamma = \mathsf{Id}$ as above, $f(\alpha, \beta) = \beta$ and $g(\alpha, \beta) = \alpha$ as before, but put $h(\alpha, \beta) = 1 : \beta$ instead. Consequently, $\mathsf{delay} = \mathsf{FCR}(\Phi, \Psi, \Gamma)(f, g, h)$ satisfies the equations:

$$\begin{aligned}
\mathsf{delay}([\,], \beta) &= f([\,], \beta) = \beta \\
\mathsf{delay}(0 : \alpha, \beta) &\simeq \mathsf{delay}(\alpha, \beta) \\
\mathsf{delay}(1 : \alpha, \beta) &\simeq \mathsf{delay}(\alpha, 1 : \beta).
\end{aligned}$$

This presents delay as an alternative version of the coinductive addition, defined by $a(0, m) = m$ and $a(n + 1, m) = a(n, 1 + m)$. In functional programming terms, add is a little more lazy, as for any infinite sequence α, we have $\mathsf{add}(\alpha, \beta) = \alpha$ whereas $\mathsf{delay}(\alpha, \beta) = 0^\omega$.

The example above is rather trivial as the function parameters are not changing in recursive calls. We had $\Phi_i(f_i) = f_i$ for all the functionals. The next example makes use of changing parameters.

▶ **Example 10.** Consider the functionals Φ, Ψ, Γ where $\Phi(f)(\alpha, \beta) = \beta$ and Ψ, Γ are arbitrary. Then, for $g(\alpha, \beta) = [\,]$ and $h(\alpha, \beta) = f(\alpha, \beta) = \alpha$ we have

$$\begin{aligned}
\mathsf{FCR}(\Phi, \Psi, \Gamma)(f, g, h)([\,], \beta) &= f([\,], \beta) = [\,] \\
\mathsf{FCR}(\Phi, \Psi, \Gamma)(f, g, h)(1 : \alpha, \beta) &\simeq \mathsf{FCR}(\Phi, \Psi, \Gamma)(f', g', h')(g(\alpha, \beta), h(\alpha, \beta)) \\
&= \mathsf{FCR}(\Phi, \Psi, \Gamma)(f', g', h')([\,], \alpha) \\
&= f'([\,], \alpha) = \alpha
\end{aligned}$$

where $f' = \Phi(f)$, $g' = \Psi(g)$ and $h' = \Gamma(h)$.

In other words, $\mathsf{pred}(\alpha) = \mathsf{FCR}(\Phi, \Psi, \Gamma)(f, g, h)(\alpha, [\,])$ is the predecessor function that removes the first occurrence of 1 from α if it exists. It is intriguing to note that the complexity of pred, unlike its primitive recursive counterpart, is not linear but instead depends on the representation of α.

▶ **Example 11.** Let $f : \overline{\mathbb{N}}^{k+1} \to \overline{\mathbb{N}}$ be extensional. Consider the $k+2$ functionals $\vec{\Phi} = (\Phi_0, \Phi_1, \ldots, \Phi_{k+1})$ and functions $\vec{f} = (f_0, f_1, \ldots f_{k+1})$ such that:

$$f_0(\vec{\alpha}) = \alpha_2 \qquad f_1(\vec{\alpha}) = f(1 : \alpha_2, \vec{\alpha}) \qquad f_2(\vec{\alpha}) = 1 : \alpha_2 \qquad f_i(\vec{\alpha}) = \alpha_i \text{ for } i > 2$$

where $\vec{\alpha} = (\alpha_1, \ldots, \alpha_{k+1})$ and $\Phi_i(f_i) = f_i$ for $i = 0, \ldots, k+1$.
It follows that:

$$\begin{aligned}
&\mathsf{FCR}\,\vec{\Phi}\,\vec{f}(f([\,], \vec{\alpha}), [\,], \vec{\alpha}) \simeq [\,]\ \underline{\mathsf{r}}\ 0 && \text{if } f([\,], \vec{\alpha})\ \underline{\mathsf{r}}\ 0 \\
&\mathsf{FCR}\,\vec{\Phi}\,\vec{f}(f([\,], \vec{\alpha}), [\,], \vec{\alpha}) \simeq \mathsf{FCR}\,\vec{\Phi}\,\vec{f}(f([1], \vec{\alpha}), [1], \vec{\alpha}) && \text{otherwise}
\end{aligned}$$

Continuing in the same way, we have:

$$\begin{aligned}
&\mathsf{FCR}\,\vec{\Phi}\,\vec{f}(f([1], \vec{\alpha}), [1], \vec{\alpha})\ \underline{\mathsf{r}}\ 1 && \text{if } f([1], \vec{\alpha})\ \underline{\mathsf{r}}\ 0 \\
&\mathsf{FCR}\,\vec{\Phi}\,\vec{f}(f([1], \vec{\alpha}), [1], \vec{\alpha}) \simeq \mathsf{FCR}\,\vec{\Phi}\,\vec{f}(f([1,1], \vec{\alpha}), [1,1], \vec{\alpha}) && \text{otherwise}
\end{aligned}$$

By observing that α_2 is incremented at each iteration, we see and could easily prove by induction that the function

$$\mathsf{min} f(\vec{\alpha}) = \mathsf{FCR}\,\vec{\Phi}\,\vec{f}(f([\,], \vec{\alpha}), [\,], \vec{\alpha})$$

satisfies the pattern $\mathsf{min} f(\vec{\alpha}) \simeq n$ whenever:
- $f(n, \vec{\alpha})\ \underline{\mathsf{r}}\ 0$,
- $f(k, \vec{\alpha}) \downarrow$ for all $k < n$, and
- $f(k, \vec{\alpha})$ represents a non-zero value for all $k < n$.

If $f : \overline{\mathbb{N}}^{k+1} \to \overline{\mathbb{N}}$ is a representation of a possibly partial numeric function $\hat{f} : \mathbb{N}^{k+1} \rightharpoonup \mathbb{N}$, it follows that $\mathsf{min} f$ is a representation of $\mu\hat{f}$, and therefore $\mathsf{min} f$ is extensional by Proposition 8. Using isotone function space corecursion, we obtain a lazy and slightly simpler variant of minimisation, given by

$$\mathsf{min}' f(\alpha) = \mathsf{ICR}\,\vec{\Phi}\,\vec{f}(f([\,], \vec{\alpha}), [\,], \vec{\alpha})$$

where all functionals and functions are the same as above with the exception of $f_2(\vec{\alpha}) = \alpha_2$.

Our next, last, and most complex example shows how to obtain primitive recursion from function space corecursion.

▶ **Example 12.** Assume that $r : \overline{\mathbb{N}}^{k+2} \to \overline{\mathbb{N}}$ and $s : \overline{\mathbb{N}}^{k} \to \overline{\mathbb{N}}$ are given and extensional representations of total numeric functions $\hat{r} : \mathbb{N}^{k+2} \rightharpoonup \mathbb{N}$ and $\hat{s} : \mathbb{N}^{k} \rightharpoonup \mathbb{N}$. We construct a representation of $\mathsf{PR}(\hat{s}, \hat{r})$, that is, a function $\phi : \overline{\mathbb{N}}^{k+1} \to \overline{\mathbb{N}}$. We do this by defining a $(k+2)$-ary function θ by function space corecursion, and then putting $\phi(\alpha, \vec{\beta}) = \theta(\alpha, [\,], \beta)$. For the definition of θ, consider the functionals $\Phi, \Psi, \Gamma : (\overline{\mathbb{N}}^{k+2} \to \overline{\mathbb{N}}) \to (\overline{\mathbb{N}}^{k+2} \to \overline{\mathbb{N}})$ given by

$$\Phi(f)(\alpha_1, \alpha_2, \vec{\beta}) = r(\alpha_2, f([\,], \mathsf{pred}(\alpha_2), \vec{\beta}), \vec{\beta}) \qquad \Psi(g) = g \qquad \Gamma(h) = \mathsf{suc} \circ h$$

where $\mathsf{suc}(\alpha) = 1 : \alpha$ is prefixing of α with 1, and pred is the predecessor function from Example 10. Now consider the initial functions

$$f_0(\alpha_1, \alpha_2, \vec{\beta}) = s(\vec{\beta}) \qquad g_0(\alpha_1, \alpha_2, \vec{\beta}) = \alpha_1 \qquad h_0(\_) = [\,]$$

and let $\theta = \mathsf{FCR}(\Phi, \Psi, \Gamma, \vec{\mathsf{Id}})(f_0, g_0, h_0, \vec{\mathsf{id}})$. If $f_{n+1} = \Phi(f_n)$, $g_{n+1} = \Psi(g_n)$ and $h_{n+1} = \Gamma(h_n)$, whereas $\vec{\mathsf{id}}$ and $\vec{\mathsf{Id}}$ are sequences of k identity functions and functionals, we have $\phi(\alpha, \vec{\beta}) = \mathsf{FCR}(\Phi, \Psi, \Gamma, \vec{\mathsf{Id}})(f_0, g_0, h_0, \vec{\mathsf{id}})(\alpha, [\,], \vec{\beta})$:

$$\begin{aligned}
\phi([\,],\beta) &= \mathsf{FCR}(\Phi,\Psi,\Gamma,\vec{\mathsf{Id}})(f_0,g_0,h_0,\vec{\mathsf{id}})([\,],[\,],\vec{\beta}) = f_0([\,],[\,],\vec{\beta}) = s(\vec{\beta})\\
\phi(\mathsf{suc}\ \alpha,\vec{\beta}) &= \mathsf{FCR}(\Phi,\Psi,\Gamma,\vec{\mathsf{Id}})(f_0,g_0,h_0,\vec{\mathsf{id}})(\mathsf{suc}\ \alpha,[\,],\vec{\beta})\\
&\simeq \mathsf{FCR}(\Phi,\Psi,\Gamma,\vec{\mathsf{Id}})(f_1,g_1,h_1,\vec{\mathsf{id}})(\alpha,[\,],\vec{\beta}) \simeq \ldots \simeq\\
&\simeq \mathsf{FCR}(\Phi,\Psi,\Gamma,\vec{\mathsf{Id}})(f_{n+1},g_{n+1},h_{n+1},\vec{\mathsf{id}})([\,],\alpha,\vec{\beta})\\
&= f_{n+1}([\,],\alpha,\vec{\beta})\\
&= r(n,f_n([\,],\mathsf{pred}(\alpha),\beta),\beta) = r(\alpha,\phi(\alpha,\beta),\beta).
\end{aligned}$$

In other words, ϕ satisfies the equations of the function $\mathsf{PR}(\hat{s},\hat{r})$ defined by primitive recursion, i.e., $\phi([\,],\vec{\beta}) = s(\vec{\beta})$ and $\phi(\mathsf{suc}(\alpha),\vec{\beta}) = r(\alpha,\phi(\alpha,\vec{\beta}),\vec{\beta})$. That is, the function $\mathsf{FCR}(\Phi,\Psi,\Gamma,\vec{\mathsf{id}})(f,g,h,\vec{\mathsf{id}})(\alpha,[\,],\vec{\beta})$ is a representation of $\mathsf{PR}(\hat{s},\hat{r})$,

In the example above, we obtain a representation of $\mathsf{PR}(\hat{s},\hat{r})$ as we have assumed that $\hat{r}$ and $\hat{s}$ are *total*. For partial functions, this is not necessarily true as $f \circ g$ does not in general represent $\hat{f} \circ \hat{g}$, even if f and g represent $\hat{f}$ and $\hat{g}$, respectively. We return to this when we show that function space corecursion indeed defines all computable functions.

5 Computable Corecursive Functions

The function space corecursion operator offers a lot of flexibility for how the functionals and functions are constructed. To define the class of computable corecursive functions, we need to be precise about how, and in particular with what functionals, the operator is used. Similar to partial recursive functions, computable corecursive functions are an inductively defined class, and every element of this class denotes a k-ary function of type $\bar{\mathbb{N}}^k \to \bar{\mathbb{N}}$. As for primitive recursive functions, we often identify a k-ary function symbol with an actual function. The challenge in the definition of computable corecursive functions is precisely which functionals to permit in the definition of function space corecursion. An operator is permissible if it can be "written" in terms of its arguments and previously defined computable corecursive functions.

The definition of computable corecursive functions makes this precise by representing the functionals Φ in the definition of FCR by terms. To define a functional $\Phi : (\bar{\mathbb{N}}^k \to \bar{\mathbb{N}}) \to (\bar{\mathbb{N}}^k \to \bar{\mathbb{N}})$, we need to specify $\Phi(\phi)$ for $\phi : \bar{\mathbb{N}}^k \to \bar{\mathbb{N}}$, and to define $\Phi(\phi)$ we need to prescribe $\Phi(\phi)(\vec{\alpha})$ for a generic $\vec{\alpha} \in \bar{\mathbb{N}}^k$. In other words, we stipulate that $\Phi(\phi)(\vec{\alpha})$ is an expression that uses already defined functions, the function ϕ and the arguments $\alpha_1,\ldots,\alpha_k$. That is, we can represent $\Phi(\phi)(\vec{\alpha})$, and hence Φ itself, by a term in k variables over a signature that contains symbols for already defined computable corecursive functions, plus an extra k-ary symbol ϕ.

▶ **Definition 13** (Computable corecursive functions). *The class of* computable corecursive functions *is the least class that contains the constant zero function* 0 *of arity* 0*, the successor function* suc *of arity* 1*, the projection functions* π^k_i *of arity* k *for* $1 \le i \le k$ *and is closed under*

function composition, *i.e., if f is n-ary and $g_1,\ldots g_n$ are k-ary, and all of $f,g_1,\ldots,g_n$ are computable corecursive, then so is $f \circ \langle g_1,\ldots,g_n\rangle$, and*

function space corecursion, *that is, if Σ is a set of computable corecursive functions and $\Phi_0,\ldots,\Phi_k \in T_{\Sigma'}(\{x_1,\ldots,x_k\})$, then $\mathsf{FCR}(\Phi_0,\ldots,\Phi_k)(f_0,\ldots,f_k)$ is a computable corecursive function of arity k, provided all of $f_0,\ldots,f_k$ are k-ary and computably corecursive, and $\Sigma' = \Sigma \cup \{\phi\}$ where $\phi \notin \Sigma$ is a fresh k-ary function symbol.*

Every k-ary computable corecursive function f is a term that we often identify with its denotation $[\![f]\!] : \overline{\mathbb{N}}^k \to \overline{\mathbb{N}}$ given as follows. We have $[\![0]\!] = [\,]$, $[\![\mathsf{suc}]\!](\alpha) = 1 : \alpha$, $[\![\pi^k_i]\!](\alpha_1, \dots, \alpha_k) = \alpha_i$ and $[\![f \circ \langle g_1, \dots, g_k\rangle]\!](\alpha_1, \dots, \alpha_n) = [\![f]\!]([\![g_1]\!](\alpha_1, \dots, \alpha_n), \dots, [\![g_k]\!](\alpha_1, \dots, \alpha_k))$. For function space corecursion, the denotation of $\Phi \in T_{\Sigma'}(x_1, \dots, x_k)$ is given in the standard way by induction on the structure of Φ, that is

$$[\![\Phi]\!](g)(\vec{\alpha}) = \begin{cases} \alpha_i, & \Phi = x_i \text{ is a variable} \\ f([\![\Phi_1]\!](g)(\vec{\alpha}), \dots, [\![\Phi_n]\!](g)(\vec{\alpha})), & \Phi = f(\Phi_1, \dots, \Phi_n) \text{ and } f \in \Sigma \\ g([\![\Phi_1]\!](g)(\vec{\alpha}), \dots, [\![\Phi_n]\!](g)(\vec{\alpha})), & \Phi = \phi(\Phi_1, \dots, \Phi_k) \text{ and } \phi \in \Sigma' \setminus \Sigma \end{cases}$$

where we use $f \in \Sigma$ homonymously as a function symbol and its interpretation, $g : \overline{\mathbb{N}}^k \to \overline{\mathbb{N}}$ and $\vec{\alpha} = \alpha_1, \dots, \alpha_k \in \overline{\mathbb{N}}^k$. With this,

$$[\![\mathsf{FCR}(\Phi_0, \dots, \Phi_k)(f_0, \dots, f_k)]\!] = \mathsf{FCR}([\![\Phi_0]\!], \dots, [\![\Phi_k]\!])(f_0, \dots, f_k)$$

where function space corecursion is the same as in Definition 5.

When requiring that $\Phi \in T_{\Sigma'}(\{x_1, \dots, x_k\})$, we use already defined, n-ary computable corecursive functions $f \in \Sigma$ as n-ary function symbols. Unfolding the definition of the term algebra $T_{\Sigma'}(\{x_1, \dots, x_k\})$, we have that Φ is either a variable, or Φ is of the form $\sigma(\Phi_1, \dots, \Phi_n)$ where $\sigma \in \Sigma'$ is a function symbol. All Φ_i are already defined terms $\Phi_i \in T_{\Sigma'}(\{x_1, \dots, x_n\})$. When interpreting Φ as a functional $[\![\Phi]\!] : (\overline{\mathbb{N}}^k \to \overline{\mathbb{N}}) \to (\overline{\mathbb{N}}^k \to \overline{\mathbb{N}})$, we interpret a function symbol $\phi \in \Sigma' \setminus \Sigma$ as the argument of the interpretation of Φ, and a function symbol $f \in \Sigma$ as f (conceived as a function).

We have seen that function space corecursion can simulate both minimisation and primitive recursion. This does not suffice to show that every partial recursive function can be represented by a computable corecursive function – the composition of partial recursive functions differs from the composition of computable corecursive functions as the latter are total. Consider as an example the constant zero function $z(\alpha) = [\,]$ that represents the function $\hat{z}(n) = 0$. The unary function $f(\alpha) = (0, 0, \dots)$ represents the totally undefined function $\hat{f}$ where $\hat{f}(n)\uparrow$ for all $n \in \mathbb{N}$. It is however not the case that $z \circ f$ represents $\hat{z} \circ \hat{f}$ as $z \circ f([\,])\downarrow$ but $\hat{z} \circ \hat{f}(0)\uparrow$.

To establish a representation theorem, we therefore need to make the somewhat unnatural manoeuvre, which deliberately limits the definedness of a function, to represent the composition of partial functions using function space corecursion. The key observation is that the constant zero function $\hat{z}$ also has a different representation z' where $z'(\alpha) \simeq 0$ for any finite sequence $\alpha \in 2^*$ and $z'(\alpha) = (0, 0, \dots)$ for any infinite sequence $\alpha \in 2^\omega$. We start by showing that z' is computable corecursive.

▶ **Lemma 14.** *The function $z' : \overline{\mathbb{N}} \to \overline{\mathbb{N}}$ defined by $z'(\alpha) = 0^n$ if $\alpha \in 2^n$ and $z'(\alpha) = 0^\omega$ otherwise is computable corecursive.*

Proof. Let $z' = \mathsf{FCR}(\mathsf{Id}, \mathsf{Id})(\mathsf{id}, \mathsf{id})$ where $\mathsf{Id} : (\overline{\mathbb{N}} \to \overline{\mathbb{N}}) \to (\overline{\mathbb{N}} \to \overline{\mathbb{N}})$ is the identity functional, and $\mathsf{id} : \overline{\mathbb{N}} \to \overline{\mathbb{N}}$ is the identity function. Then

$$\begin{aligned} z'([\,]) &= \mathsf{FCR}(\mathsf{Id}, \mathsf{Id})(\mathsf{id}, \mathsf{id})([\,]) &&= [\,] \\ z'(0 : \alpha) &= \mathsf{FCR}(\mathsf{Id}, \mathsf{Id})(\mathsf{id}, \mathsf{id})(0 : \alpha) &&= 0 : \mathsf{FCR}(\mathsf{Id}, \mathsf{Id})(\mathsf{id}, \mathsf{id})(\alpha) \\ z'(1 : \alpha) &= \mathsf{FCR}(\mathsf{Id}, \mathsf{Id})(\mathsf{id}, \mathsf{id})(1 : \alpha) &&= 0 : \mathsf{FCR}(\mathsf{Id}, \mathsf{Id})(\mathsf{id}, \mathsf{id})(\alpha). \end{aligned}$$

It is evident that z' is the denotation of a computable corecursive function, and we have $z' = [\![\mathsf{FCR}(\mathsf{Id}, \mathsf{Id})(\mathsf{id}, \mathsf{id})]\!]$ where formally $\mathsf{Id} = \phi(x) \in T_{\emptyset'}(\{x\})$ is the required term representation, and id is the evidently computable corecursive identity function π^1_1. ◀

Our strategy, to show that the composition of partial recursive functions can again be represented by a computably corecursive function, is to prefix the concatenation $z'(g_1(\vec{\alpha})) \cdot \cdots \cdot z'(g_n(\vec{\alpha}))$ to the composition $f \circ \langle g_1, \ldots, g_n \rangle$ and so making sure that the composition is undefined (i.e., an infinite sequence) if one of the arguments is undefined.

▶ **Lemma 15.** *Suppose that $f : \overline{\mathbb{N}}^n \to \overline{\mathbb{N}}$ and $g_1, \ldots, g_n : \overline{\mathbb{N}}^k \to \overline{\mathbb{N}}$ are computably corecursive. If f represents $\hat{f}$ and g_i represents $\hat{g}_i$, then there is a computably corecursive function h that represents $\hat{f} \circ \langle \hat{g}_1, \ldots, \hat{g}_n \rangle$.*

Proof. We write $\alpha \cdot \beta$ for $\mathsf{delay}(\alpha, \beta)$ and put $h(\alpha_1, \ldots, \alpha_n) = z'(g_1(\vec{\alpha}) \cdot \cdots \cdot g_n(\vec{\alpha})) \cdot f(g_1(\vec{\alpha}), \ldots, g_n(\vec{\alpha}))$. Here, z' is as in Lemma 14 and it is clear that h represents $\hat{f} \circ \langle g_1, \ldots, g_k \rangle$. ◀

We have now collected all the ingredients for our representation theorem.

▶ **Theorem 16.** *Every partial recursive function has a computable corecursive representation.*

Proof. For total functions $\hat{f}$ and $\hat{g}_1, \ldots, \hat{g}_n$ that are represented by $f, g_1, \ldots, g_n$, we have that $f \circ \langle g_1, \ldots, g_n \rangle$ represents $\hat{f} \circ \langle \hat{g}_1, \ldots \hat{g}_n \rangle$. It then follows from Example 12 that all primitive recursive functions have a representation. To obtain a representation for all partial recursive functions, we use Kleene's normal form theorem and Example 11 where we implement the composition of partial functions using Lemma 15. ◀

6 Soundness

We have shown that every partial recursive function is represented by the denotation of a computable corecursive function. We could argue that this implies computable corecursive functions represent the class of all computable functions by Church's thesis, as they are intuitively computable. Indeed, it is obvious how we implement an instance of function space corecursion in the Haskell programming language. The following shows an example of how FCR can be defined.

```
-- finite and infinite lists
data Bit = Zero | One deriving (Eq, Show); type Seq = [Bit]

-- functions and functionals for an instance of FCR with two arguments
type Fn = Seq -> Seq -> Seq
type Fnl = Fn -> Fn

-- function space corecursion for binary functions
fcr2 :: Fnl -> Fnl -> Fnl -> Fn -> Fn -> Fn -> Fn
fcr2 phi psi gamma f g h a b = case a of
  []     -> f [] b
  Zero:a -> Zero:(fcr2 phi psi gamma f g h a b)
  One:a  -> Zero:(fcr2 phi psi gamma f' g' h' a' b') where
    f' = phi f; g' = psi g; h' = gamma h; a' = g a b; b' = h a b
```

The delay and pred functions in the previous section are then easily defined as follows.

```
-- delayed addition
delay :: Fn
delay = fcr2 id id id (\a b -> b) (\a b -> a) (\a b -> One:b)
```

```
-- auxiliary function to define predecessor
pred' :: Fn
pred' = fcr2 (\f -> \a b -> b) id id (\a b -> a) (\a b -> []) (\a b -> a)
```

While the argument is certainly convincing, we instead show that all computable corecursive functions are λ-representable, starting with the encoding of finite and infinite sequences of digits as untyped λ-terms. In addition to the binary digits, we introduce a blank symbol (B) to mark the end of a binary sequence.

▶ **Definition 17.** *Define* $[\![\cdot]\!] : 2 \cup \{B\} \to \Lambda$ *such that* $[\![B]\!] = \lambda xyz.x$, $[\![0]\!] = \lambda xyz.y$ *and* $[\![1]\!] = \lambda xyz.z$. *A term* M *encodes a finite binary sequence* $\alpha = (a_0, \dots, a_{k-1})$ *whenever* $M\ \lceil i \rceil = [\![a_i]\!]$ *for all* $0 \le i < k$, *and* $M\ \lceil i+1 \rceil = [\![B]\!]$ *for* $i = k$. *A term* M *encodes an infinite binary sequence* $(\alpha_0, \alpha_1, \dots)$ *whenever* $M\ \lceil i \rceil = [\![a_i]\!]$ *for all* $i \in \mathbb{N}$.

As mentioned previously, $\lceil \cdot \rceil : \mathbb{N} \to \Lambda$ is a mapping from natural numbers to their λ-encoding. Barendregt's formulation [3] or any alternative encoding that provides basic functionalities (e.g., predecessor and successor functions) works the same way.

▶ **Example 18.** The empty sequence $[\,]$ can be encoded as $\lambda x.[\![B]\!]$ because $(\lambda x.[\![B]\!])\ \lceil 0 \rceil \longrightarrow_\beta [\![B]\!]$. An infinite sequence of 1's can be encoded as $\lambda x.[\![1]\!]$. In fact, we can define a concatenation operator cons by

$$\mathsf{cons}\ x\ y \equiv \lambda w.\mathsf{if}\ (\mathsf{isZero}\ w)\ \mathsf{then}\ x\ \mathsf{else}\ y\ (\mathsf{pred}\ w)$$

where isZero checks whether w encodes the natural number zero, and pred decrements a natural number. The head and tail functions are simply $\lambda x.x\ \lceil 0 \rceil$ and $\lambda xy.x\ (\mathsf{suc}\ y)$, respectively.

To facilitate case distinctions with the binary digits and the blank symbol, let the notation $\mathsf{switch}\ W\ \mathsf{caseB}\ X\ \mathsf{case0}\ Y\ \mathsf{case1}\ Z$ be the syntactic sugar of $W\ X\ Y\ Z$. The encoding of the blank symbol and the binary digits effectively function as truth values for the case distinction. For example,

$$\begin{aligned} &\mathsf{switch}\ [\![B]\!]\ \mathsf{caseB}\ X\ \mathsf{case0}\ Y\ \mathsf{case1}\ Z \\ \equiv\ &[\![B]\!]\ X\ Y\ Z \\ \longrightarrow_\beta^*\ &(\lambda xyz.x)\ X\ Y\ Z \\ \longrightarrow_\beta^*\ &X. \end{aligned}$$

▶ **Definition 19** (λ-definability of functions)**.** *A computable corecursive function* $f : \overline{\mathbb{N}}^k \to \overline{\mathbb{N}}$ *is* λ-definable *if there exists* $F \in \Lambda$ *such that* $F\ R_1\ \dots\ R_k$ *is the encoding of* $f(\alpha_1, \dots, \alpha_k)$ *as per Definition 17 whenever* R_i *is a* λ*-encoding of* α_i. *A functional* $\Phi : (\overline{\mathbb{N}}^k \to \overline{\mathbb{N}}) \to (\overline{\mathbb{N}}^k \to \overline{\mathbb{N}})$ *is* λ*-definable if there exists* $P \in \Lambda$ *such that* $P\ F$ *is an encoding of* $\Phi(f)$ *whenever* F *is the encoding of* $f : \overline{\mathbb{N}}^k \to \overline{\mathbb{N}}$.

▶ **Example 20.** If functionals $\Phi_1, \dots, \Phi_k$ and functions $f_1, \dots, f_k$ are λ-definable, then so is $\mathsf{FCR}(\Phi_0, \dots, \Phi_k)(f_0, \dots, f_k)$. Suppose $P_1, \dots, P_k$ are the λ-encodings of the functionals $\Phi_0, \dots, \Phi_k$, and $F_0, \dots, F_k$ are the encodings of the functions $f_0, \dots, f_k$. Devise F as

$$\begin{aligned} &\lambda f.P_0 \cdots P_k\ F_0 \cdots F_k\ R_1 \cdots R_k. \\ &\quad \mathsf{switch}\ (\mathsf{head}\ R_1) \\ &\quad \mathsf{caseB}\ (f\ F_0\ R_1 \cdots R_k) \\ &\quad \mathsf{case0}\ (\mathsf{cons}\ [\![0]\!]\ (f\ P_0 \cdots P_k\ F_0 \cdots F_k\ (\mathsf{tail}\ R_1)\ R_2 \cdots R_k)) \\ &\quad \mathsf{case1}\ (\mathsf{cons}\ [\![0]\!]\ (f\ P_0 \dots P_k\ F_0' \cdots F_k'\ (F_1\ (\mathsf{tail}\ R_1))\ R_2' \cdots R_k')) \end{aligned}$$

where $F_i' \equiv P_i\ F_i$ and $R_i' \equiv F_i\ R_i$. It follows that $Y\ F$ is the encoding of the function $\mathsf{FCR}(\Phi_0, \dots, \Phi_k)(f_0, \dots, f_k)$ with Y being the Y-combinator.

▶ **Theorem 21.** *All computable corecursive functions are λ-definable.*

Proof. By induction on the class of computable corecursive functions, replacing the interpretation $[\![\cdot]\!]$ in Definition 5 with $[\![\cdot]\!]_\lambda$ that maps the terms to their λ-encodings. The base cases are evident while the encoding of FCR is shown in Example 20. ◀

7 Conclusions and Further Work

We have seen how function space corecursion and its variants can be used to represent exactly the class of all computable functions. This (co)algebraic approach offers a mathematically elegant way to represent computer programs. This computation model avoids the intricacies of state transitions and side effects such as tape manipulations found in Turing machines; it also enables certain programming features (e.g., lazy evaluation) that cannot be reflected in partial recursive functions to be introduced. Mixed induction-coinduction has been used for interpreting the coinductive data structure of binary sequences, providing extensionality and allowing us to correlate the data structure to program non-termination.

The major result of this paper has been formally investigating the (co)algebraic foundations of the model, along with proving its expressive power in relation to existing models of computation. It is envisioned to open a number of intriguing research perspectives.

The first open question is, can we establish the standard results of recursion theory also with computable corecursive functions? For example, if ϕ is an enumeration of all computable corecursive functions and f is total and recursive, do we have e such that $\phi_e \simeq \phi_{f(e)}$? The main difference from the standard formulation of the recursion theorem is that for computable corecursive functions, ϕ_e and $\phi_{f(e)}$ also need to agree on all infinite values (modulo extensional equality $\simeq$) as $\simeq$ does not identify all infinite sequences. Similar comments apply to the second recursion theorem and to the *s-m-n* theorem. Does the normal form theorem extend to computable corecursive functions?

The second intriguing avenue is to explore an equational calculus for asserting extensional equality of computable corecursive functions. Given that the denotation of computable corecursive functions are total functions on intensional natural numbers, the main barrier for the design of an equational calculus has been removed. On the other hand, equality between computable functions is not recursively enumerable. We, therefore, expect this calculus to be non-wellfounded, similar to the exercise conducted for Turing machines [14]. This question is closely related to that of induction principles for intensional natural numbers that we would employ in a calculus for equality. In other words, how can we show that $f(\alpha) \simeq g(\alpha)$ for all intensional natural numbers α? We hypothesise that any principle will combine induction and coinduction with reference to comparable systems [5].

Another orthogonal angle is that of the computational complexity of a recursive function. Can we establish a linkage between the complexity of a function f, that is, the number of steps it takes to evaluate $f(\alpha)$, with the length of the representation of $f(\alpha)$? To do this, we would move from a unary to a binary coding of natural numbers, and still retain a hiaton, i.e., a token that symbolises a computational step that has not (yet) yielded an observable result. Are there definition principles that relate the length $|f(\alpha)|$ to the number of steps it takes to compute $f(\alpha)$? Partial recursive functions are rarely used for computational complexity analysis; we envision computable corecursive functions can bridge this gap.

Finally, what is the "right" set of computable functions on 2^∞? In this paper, we have identified a class of functions on 2^∞ that represents all partial recursive functions. We have shown that for this purpose, the function space corecursion operator FCR suffices. This operator is special, for example, when representing μf using FCR, no digits of outputs are produced until a zero of f has been found, as we have discussed in Example 11. A small set of functions based on functional space corecursion may suffice to achieve Turing completeness, and thus further simplify the foundational theory.

References

1 Samson Abramsky and C.-H. Luke Ong. Full abstraction in the lazy lambda calculus. *Inf. Comput.*, 105(2):159–267, 1993. doi:10.1006/inco.1993.1044.

2 Robert Atkey and Conor McBride. Productive coprogramming with guarded recursion. *ACM SIGPLAN Notices*, 48(9):197–208, 2013. doi:10.1145/2500365.2500597.

3 Henk Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1985.

4 Michael Barr and Charles Wells. *Category Theory for Computing Science*. Prentice Hall, 1989.

5 Henning Basold and Helle Hvid Hansen. Well-definedness and observational equivalence for inductive-coinductive programs. *Journal of Logic and Computation*, 29(4):419–468, 2019. doi:10.1093/logcom/exv091.

6 George Mark Bergman. *An Invitation to General Algebra and Universal Constructions*. Universitext. Springer, Cham, 2nd edition, 2015. doi:10.1007/978-3-319-11478-1.

7 Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Foundational extensible corecursion: a proof assistant perspective. In Kathleen Fisher and John Reppy, editors, *Proc. ICFP 2015*, pages 192–204. Association for Computing Machinery, 2015. doi:10.1145/2784731.2784732.

8 Ana Bove, Alexander Krauss, and Matthieu Sozeau. Partiality and recursion in interactive theorem provers — an overview. *Mathematical Structures in Computer Science*, 26(1):38–88, 2014. doi:10.1017/S0960129514000115.

9 Martín Hötzel Escardó. On lazy natural numbers with applications to computability theory and functional programming. *SIGACT News*, 24(1):61–67, 1993. doi:10.1145/152992.153008.

10 Joel David Hamkins. A survey of infinite time turing machines. In Jérôme Olivier Durand-Lose and Maurice Margenstern, editors, *Proc. MCU 2007*, volume 4664 of *Lecture Notes in Computer Science*, pages 62–71. Springer, 2007. doi:10.1007/978-3-540-74593-8_5.

11 Bart Jacobs and Jan Rutten. A Tutorial on (Co)Algebras and (Co)Induction. *EATCS Bulletin*, 62:222–259, 1997.

12 Dexter Kozen. *Theory of Computation*. Texts in Computer Science. Springer, London, 2006. doi:10.1007/1-84628-477-5_7.

13 Rasmus Ejlers Møgelberg. A type theory for productive coprogramming via guarded recursion. In *Proc. CSL-LICS 2014*, pages 1–10, New York, 2014. Association for Computing Machinery. doi:10.1145/2603088.2603132.

14 Dirk Pattinson and Lutz Schröder. Program Equivalence is Coinductive. In *Proc. LICS 2016*, pages 337–346, New York, 2016. Association for Computing Machinery. doi:10.1145/2933575.2934506.

15 Anton Setzer. Partial Recursive Functions in Martin-Löf Type Theory. In *Proc. CiE 2006*, volume 3988 of *Lecture Notes in Computer Science*, pages 505–515. Springer, 2006. doi:10.1007/11780342_51.

16 Joseph Robert Shoenfield. *Recursion Theory*. Lecture Notes in Logic. Springer, 1993.

17 Tarmo Uustalu and Varmo Vene. Primitive (Co)Recursion and Course of Value (Co)Iteration, Categorically. *INFORMATICA (IMI, Lithuania)*, 10:5–26, 1999. doi:10.3233/INF-1999-10102.

18 Wolfgang Wechler. *Universal Algebra for Computer Scientists*, volume 25 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1992. doi:10.1007/978-3-642-76771-5.

19 Klaus Weihrauch. *Computable Analysis*. Springer, 2000.

20 Glynn Winskel. *The formal semantics of programming languages: an introduction*. MIT Press, 1993.